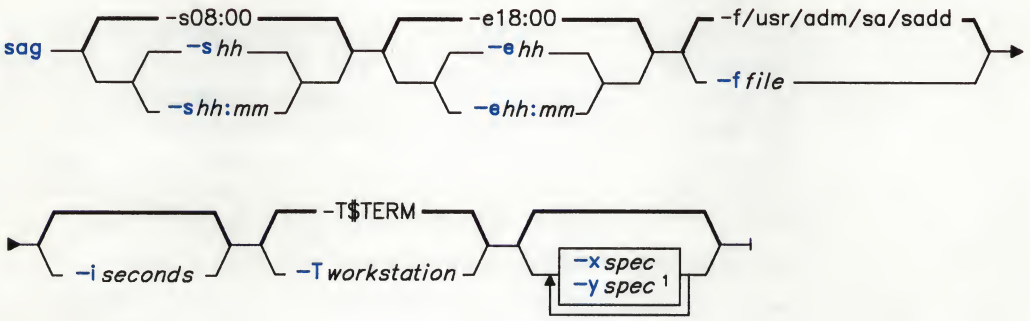
# sag

## Purpose

Displays a graph of system activity.

## Syntax




[1] The default for -y is '% usr 0 100; % usr + % sys 0 100; % usr + % sys + %wio 0 100'

OL805387

## Description

The **sag** command displays a graph of system activity. It gets information either from the daily activity file **usr/adm/sa/sa***dd* or from the binary data file selected by the **-f** flag. You must have already created this file by running the **sar** command with the **-o** flag. (See "**sar**" on page 867.)

The **sag** command calls the **sar** command, selecting the desired data by string-matching the data column header.

## Flags

The **sag** command passes the first four of the following flags to **sar** in order to collect the desired data for display. The last three flags specify plotting parameters.

**-e** *hh*[*:mm*] Selects data up to the time specified by *hh*[*:mm*]. The default time is 18:00.

**-f** *file* Reads data from *file*. The default *file* is **/usr/adm/sa/sa***dd*, the current daily data file.

**-i** *seconds* Selects data at intervals as close as possible to *seconds*.

**-s** *hh*[**:***mm*] Selects data later than the specified time. Default is 08:00.

**-T** *workstation* Produces output suitable for *workstation*. (See "**tplot**" on page 1079 for known work stations.) If you do not specify a work station, **sag** uses the value found in the shell variable **$TERM**.

**-x** *spec* Specifies the x axis. *spec* has the following form:

*name* [*opname*] . . . [*lo hi*]

where *name* is a character string matching a column header in the **sar**-created data file (with an optional device name in brackets), or it is an integer value. *op* is +, -, *, or / surrounded by blanks, with up to five *name*s specified. Parentheses are not recognized and evaluation is left to right. Note that + and - have precedence over * and / in evaluating expressions. *lo* and *hi* specify numeric scale limits. If these limits are unspecified, **sag** gets these limits from the data.

**-y** *spec* Specifies the y axis. *spec* has the same form as **x** *spec*.

Specify only one *spec* for the x axis. If unspecified, the x axis assumes the time specified with the **-e** and **-s** flags (or their defaults if they are not used) as x axis limits. You can specify up to five *spec*s separated by : (semicolons) for **-y**. If unspecified, the y axis has the value:

```
-y "%usr 0 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"
```

If you include blanks or an escaped carriage return (**Enter**) within the **-x** and **-y** *spec*s, enclose them in " " (double quotation marks).

## Files

/usr/adm/sa/sa*dd* Daily data file for day *dd*.

## Related Information

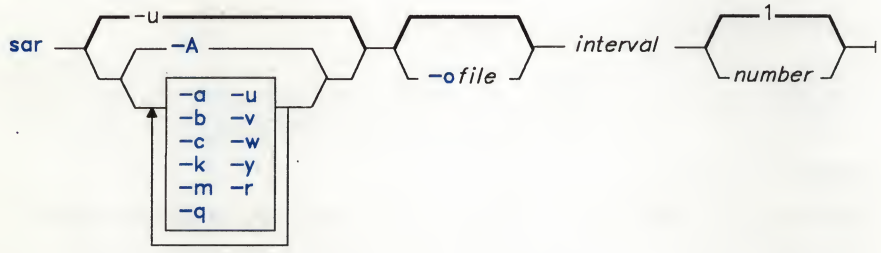
The following commands: "**sar**" on page 867 and "**tplot**" on page 1079.

# sar

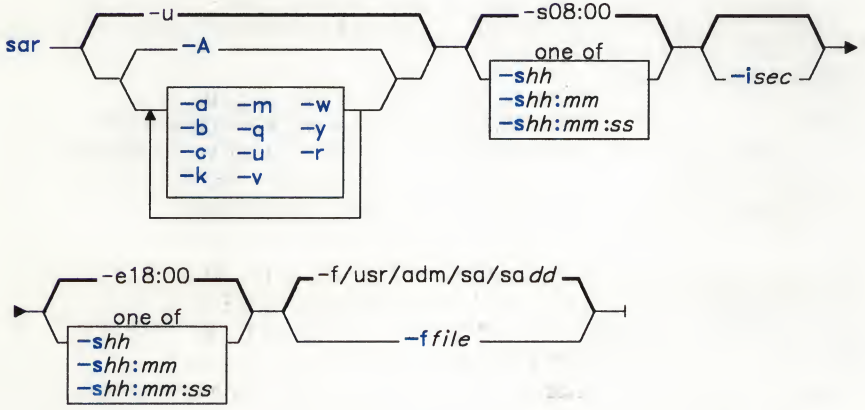
## Purpose

Collects, reports, or saves system activity information.

## Syntax




OL805390




OL805369

## Description

The first format of the **sar** command writes to standard output the contents of selected cumulative activity counters in the operating system. It writes information a total of *number* times spaced *interval* seconds apart. The default value of *number* is 1. You can also save the collected data in the file specified by **-o** *file*.

In the second format (with no sampling interval specified), **sar** extracts and writes to standard output records previously saved in a file. This file can be either the one specified by the **-f** flag or, by default, the standard system activity daily data file, **/usr/adm/sa/sa***dd*, for the current day, *dd*.

You can select information about specific system activities with flags. Not specifying any flags selects only **cpu** activity. Specifying the **-A** flag selects all activities.

**Note:** This command only reports on local activities.

## Flags

**-a** Reports use of file access system routines:

| | |
|---|---|
| iget/s | Calls per second to the i-node look-up routine. |
| namei/s | Calls per second to the directory search routine. |
| dirblk/s | Directory blocks read per second by namei(). |

**-A** Report all data.

**-b** Reports buffer activity for transfers, accesses, and cache hit ratios:

| | |
|---|---|
| lread/s, lwrit/s | Number of logical read/write requests per interval. |
| bread/s, bwrit/s | Number of block read/write operations per interval. |
| %rcache, %wcache | Cache hit ratios (for example, 1 - bread/lread). |
| pread/s, pwrit/s | Read/writes per interval on seekable raw devices. |

**-c** Reports system calls:

| | |
|---|---|
| scall/s | Total number of system calls per second. |
| rchar/s, wchar/s | Characters transferred per interval by read/write calls. |
| sread/s, swrit/s | |
| fork/s, exec/s | Specific system calls per second. |

**-e** *hh*[:*mm*[:*ss*]]
Sets the ending time of the report. The default ending time is 18:00.

**-f** *file* Extracts records from *file* (created by **-o** *file*). The default *file* is the current daily data file, **/usr/adm/sa/sa***dd*.

**-i** *seconds* Selects data records at intervals as close as possible to the specified number of *seconds*. Otherwise, **sar** reports all intervals found in the data file.

**-k** Reports kernel activity:

| | |
|---|---|
| ksched/s | Number of kernel processes assigned to tasks per second. |
| kproc-ov/s | Number of overflows occurring between sampling points. |
| kexit/s | Number of kernel processes terminating per second. |

**-m** Reports message and semaphore activities:

| | |
|---|---|
| msg/s | IPC message primitives per second. |
| sema/s | IPC semaphore primitives per second. |

**-o** *file* Saves the readings in *file* in binary form. Each reading is in a separate record and each record contains a tag identifying the time of the reading.

**-q** Reports average queue length while occupied, and percentage of time occupied:

| | |
|---|---|
| runq-sz, %runocc | Runs queue of processes in memory and runable. |

**-r** Reports VRM paging statistics:

| | |
|---|---|
| slots | The number of free pages on the paging minidisk. |
| cycle/s | The number of page replacement cycles per second. |
| fault/s | The number of page faults per second. |
| odio/s | The number of nonpaging disk I/Os per second. |

**-s** *hh*[:*mm*[:*ss*]]
Sets the starting time of the data. That is, extract records time-tagged at or following the time specified. The default starting time is 08:00.

**-u** Reports CPU activity (this flag is on by default):

| | |
|---|---|
| %usr | Percentage of CPU time devoted to the user. |
| %sys | Percentage of CPU time devoted to the kernel. |
| %wio | Percentage of CPU time waiting for block I/O to complete. |
| %idle | Percentage of CPU time idle. |

**-v** Reports status of text, process, i-node, and file tables:

| | |
|---|---|
| text-sz, proc-sz, inod-sz, file-sz | Entries in use at each sample point for each table. |
| text-ov, proc-ov, inod-ov, file-ov | Overflows occurring at each sample point for each table. |

**-w** Reports system switching activity:

| | |
|---|---|
| pswch/s | Process switches per second. |

**-y** Reports TTY device activity:

| | |
|---|---|
| rawch/s | TTY raw input queue characters per second. |
| canch/s | TTY canonical input queue characters per second. |
| outch/s | TTY output queue characters per second. |
| revin/s | TTY receive interrupts per second. |
| xmtin/s | TTY transmit interrupts per second. |
| mdmin/s | TTY modem interrupts per second. |

## Files

| | |
|---|---|
| /usr/adm/sa/sa*dd* | Daily data file, where *dd* are numbers representing the day of the month. |

## Related Information

The following command: "**sag**" on page 865.

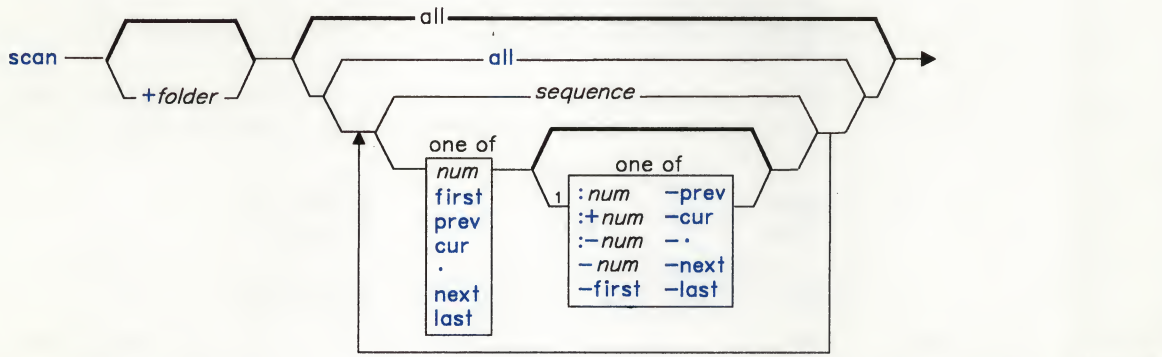
The discussion of monitoring system activity in *Managing the AIX Operating System*.

# scan

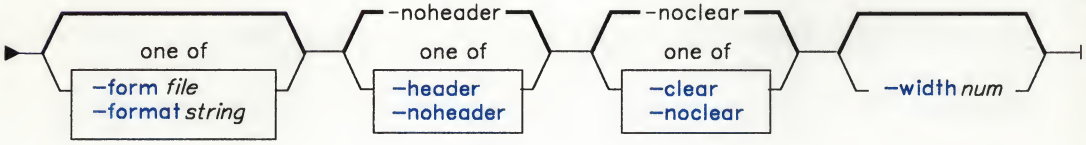
## Purpose

Produces a one line per message scan listing.

## Syntax




AJ2FL204





AJ2FL206

[1] Do not put a blank between these items.

OL805308

## Description

The **scan** command is used to display information about the messages in a specified folder. **scan** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **scan** command displays a line of information about each specified message in the specified folder. Each line gives the message number, the date, the sender, the subject, and as much of the message body as possible. If a + symbol is displayed after the message number, the message is the current message for the folder. If a - symbol is displayed, you have replied to the message. If a * symbol is displayed after the date, the **Date:** field was not present and the displayed date is the last date the message was changed.

## Flags

**-clear** Clears the display after sending output. **scan** uses the values of the **$TERM** and **$TERMCAP** environment variables to determine how to clear the display. If standard output is not a display, **scan** sends a form feed character after sending the output.

*+folder msgs* Displays information about each specified message in the specified folder. You can use the following message references when specifying *msgs*:

| | | |
|---|---|---|
| *num* | **first** | **prev** |
| **cur** | **.** | **next** |
| **last** | **all** | *sequence* |

The default folder is the current folder. If a folder is specified, it becomes the current folder. The default for *msgs* is **all**.

**-form** *file* Displays the **scan** command output in the alternate format described by *file*.

**-format** *string* Displays the **scan** command output in the alternate format described by *string*.

**-header** Displays a heading that lists the folder name and the current date and time.

**-help** Displays help information for the command.

**-noclear** Does not clear the terminal after sending output. This is the default.

**-noheader** Does not display a heading. This is the default.

**-width** *num* Sets the number of columns in the **scan** command output. The default is the width of the display.

## Profile Entries

**Alternate-Mailboxes:** Specifies your mailboxes.
**Current-Folder:** Sets your default current folder.
**Path:** Specifies your *user_mh_directory*.

## Files

| | |
|---|---|
| $HOME/.mh_profile | The MH user profile. |

## Related Information

Other MH commands: "**inc**" on page 518, "**pick**" on page 748, "**show**" on page 942.

The **mh-format** and **mh-profile** files in *AIX Operating System Technical Reference.*

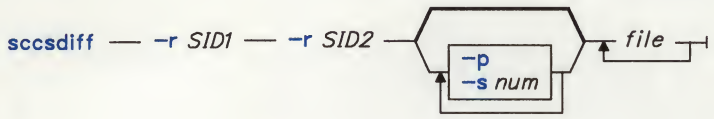
"Overview of the Message Handling Package" in *Managing the AIX Operating System.*

# sccsdiff

## Purpose

Compares two versions of a Source Code Control System (SCCS) file.

## Syntax




OL805258

## Description

The **sccsdiff** command reads two versions of an **SCCS** file, compares them, and writes to standard output the differences between the two versions. Any number of SCCS files can be specified, but the same arguments apply to all files.

## Flags

| | |
|---|---|
| **-p** | Pipes the output through **pr**. |
| **-r***SID1* | Specifies *SID1* as one delta of the SCCS file for **sccsdiff** to compare. |
| **-r***SID2* | Specifies *SID2* as the other delta of the SCCS file for **sccsdiff** to compare. |
| **-s***num* | Specifies the file segment size for **bdiff** to pass to **diff**. This is useful when **diff** fails due to a high system load. |

## Related Information

The following commands: "**bdiff**" on page 102, "**get**" on page 477, "**help**" on page 513, and"**pr**" on page 761.

The **sccsfile** file in *AIX Operating System Technical Reference*.

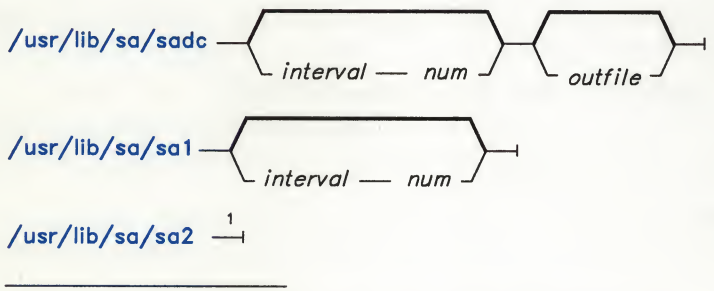
The discussion of SCCS in *AIX Operating System Programming Tools and Interfaces*.

# sadc

## Purpose

Provides a system activity report package.

## Syntax




[1] See the **sar** command for the format and flag description. Note that you cannot use the **-o** and **-f** flags with **sa2**.

OL805254

## Description

The operating system contains a number of counters that are incremented as various system actions occur. They include the following:

- System unit utilization counters
- Buffer usage counters
- Disk and tape I/O activity counters
- **tty** device activity counters
- Switching and system-call counters
- File-access counters
- Queue activity counters
- Interprocess communications counters

The **sadc** command and the **sa1** and **sa2** shell procedures sample, save, and process this data.

**Note:** These commands only report on local activities.

### sadc

The **sadc** command, the data collector, samples system data *num* times every *interval* seconds. It writes in binary format to *outfile* or to the standard output. If you do not specify *interval* or *num*, a special record is written. This facility is used at system startup to mark the time when the counter restarts from zero.

### sa1

Use the shell procedure **sa1**, a variant of **sadc** to collect and store binary data in the file **/usr/adm/sa/sa***dd*, where *dd* is the day of the month. The *interval* and *num* parameters specify that the record should be written *num* times at *interval* seconds. If you do not specify these parameters, one record is written. You must have permission to write in the directory **/usr/adm/sa** to use this command.

The **sa1** command is designed to be started automatically by the **cron** command.

---

**Japanese Language Support Information**

This command has not been modified to support Japanese characters.

---

### sa2

Use the shell procedure **sa2**, a variant of the **sar** command, to write a daily report in the file **/usr/adm/sa/sar***dd*. See "**sar**" on page 867 for a description of the flags.

The **sa2** command is designed to be started automatically by the **cron** command.

---

**Japanese Language Support Information**

This command has not been modified to support Japanese characters.

---

## Files

| | |
|---|---|
| /usr/adm/sa/sa*dd* | Daily data file, *dd* represents the day of the month. |
| /usr/adm/sa/sar*dd* | Daily report file, *dd* represents the day of the month. |
| /tmp/sa.adrfl | Address file. |

## Related Information

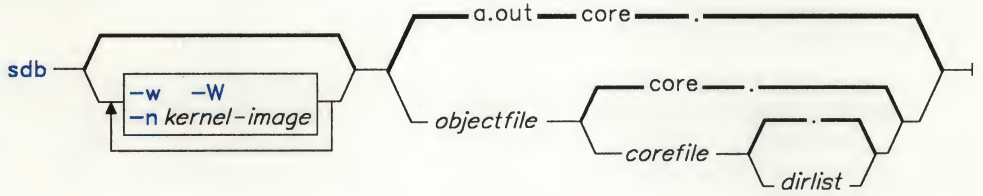
The following commands: "**cron**" on page 220, "**sag**" on page 865, "**sar**" on page 867, and "**timex**" on page 1069.

# sdb

## Purpose

Provides a symbolic debugger for C and assembler programs.

## Syntax




OL805214

## Description

The **sdb** command provides a symbolic debugger to be used with C and assembler programs. With it you can examine object and core files and provide a controlled environment for running a program. You can set breakpoints at selected statements or run the program one line at a time. You can debug using symbolic variables and instruct **sdb** to display them in their correct format.

Normally, *objectfile* is an executable file produced by invoking **cc** with the **-g** flag. If you have not compiled *objectfile* using the **-g** flag or if it is not executable (because of compiler or loader errors), the symbolic capabilities of **sdb** are limited, but you can still examine the file and debug the program. *objectfile* should always be in the same directory as the source files used to construct it. Its default name is **a.out**.

The *corefile* parameter specifies a core image file. Its default is **core**. The system writes out this core image of a process when it ends abnormally. Specifying - (minus) for *corefile* instructs **sdb** to ignore any core image file that may be present. The colon-separated list of directories specified by the *dirlist* parameter identifies the location of the source files used to build *corefile*. The default is the current directory. If *dirlist* is the name of a file, the contents of the file should be a colon-separated list of directory names.

While running, **sdb** always recognizes a ***current line*** and ***current file***. If *corefile* exists, **sdb** initially sets the current line and the current file to the line and the file that contains the source statement at which the process ended. Otherwise, it sets them to the first line in **main** and the file containing **main**. There is also a ***current function***, which is the function you are working with at any given time. You can change the current line, file, or function with the **e** command.

Write variable names as you do in C language programs. Access variables local to a function by using the form *function*:*variable*. The current function is the default function. You can also specify a variable by its address. Since you can use all forms of integer constants which are valid in C, addresses can be expressed as decimal, octal, or hexadecimal values.

Refer to structure members as *variable*.*member*, pointers to structure members as *variable*->*member*, and array elements as *variable*[*number*].

If you use the form *number*.*member* or *number*->*member*, **sdb** assumes *number* to be the address of the last structure referred to. Generally, **sdb** interprets a structure as a set of variables. Thus, it displays the values of all elements when you request it to display a structure. If, however, you request the address of a structure, it displays this value and not the addresses of individual elements.

Refer to elements of a multidimensional array as *variable*[*number*][*number*] or as *variable*[*number*,*number*]. In place of *number*, use the form *number*;*number* to indicate a range of values. You can also use an * (asterisk) to represent all legitimate values for a subscript or omit subscripts to indicate the full range of values. As with structures, **sdb** displays all the values of an array or of a section of an array if you omit trailing subscripts. If you omit subscripts, it displays only the address of the array itself or of the section specified.

Refer to a variable on the stack by using the form *function*:*variable*,. Here, *number* specifies the variable's location on the stack, counting the top, or most recently pushed variable, as the first. Use this for recursive function calls. The current function is the default.

Refer to line numbers as *filename*:*number* or *function*:*number*. The current file and current function are the default values.

**Notes:**

1. Data stored in text sections is indistinguishable from functions.
2. Line number information in optimized functions is unreliable, and some information may be missing.
3. Source line and local symbol information for routines in shared libraries is not implemented, and these modules should not be compiled with the **-g** flag. Break points may be set in these routines by address only, and code in shared library modules may be single-stepped by instruction only.
4. The **sdb** command cannot comprehend a module in which C functions (as opposed to declarations and preprocessor definitions) occur in include files.

## Flags

**-n** *kernel-image* Specifies the name of the running kernel (or the one running when *corefile* was produced). This enables proper traces back through the floating-point emulation code. **/unix** is the default value.

**-w** Allows overwriting of locations in *objectfile*.

**-W** Turns off the warnings normally given if source files cannot be found or are newer than *objectfile*.

## Subcommands

### Examining Program Data

**T** Displays the top line of the stack trace.

**t** Displays a stack trace of the program that ended abnormally.

*variable*/[*nlf*] Displays *variable* or *n* memory locations starting at the address of *variable*.

The *l* parameter selects the number of bytes in one memory location. Your choices are:

**b** One byte
**h** Two bytes
**l** Four bytes.

The *f* parameter selects the display format. This can be one of the following:

**a** Displays all bytes from the address of the *variable* to the first null byte.
**c** Displays a character value.
**d** Displays a decimal value.
**f** Displays a 32-bit, single-precision floating-point value.
**g** Displays a 64-bit, double-precision floating-point value.
**I** Interprets values as assembler language instructions and displays numerically.
**i** Interprets values as assembler language instructions and displays them numerically and symbolically.
**o** Displays an octal value.
**t** Displays F if *variable* = 0; otherwise, displays T.
**p** Displays a pointer to a function.
**s** Treats *variable* as a string pointer and displays characters beginning with the address to which it points and ending at the first null byte reached.

**u** Displays an unsigned decimal value.
**x** Displays a hexadecimal value.

If you do not specify *n*, *l*, or *f*, **sdb** chooses a value appropriate to *variable* type as declared in the source file. Specifying a memory location size works only with formats **c**, **d**, **o**, **u**, and **x**. You can specify the number of memory locations (*n*) to be displayed by the **s** or **a** formats.

For strings that contain 2-byte extended characters, the font shift character is represented by a \ (backslash) followed by lowercase **s** and the font shift number.

For example, **\s1** means that the current byte being displayed is a font shift character. This form of representation for the font shift byte is only available when a count is specified. However, if the first character contained in the address specified by the **a** format is the second byte of a 2-byte extended character, then that byte is displayed without the proper shift affixed to construct the whole 2-byte sequence. (When Japanese Language Support is installed on your system, the font shift symbols are not used. Strings can contain any combination of ASCII, 1-byte kana, and kanji characters.)

The default action for these formats is to display characters until either a null byte is reached or 128 characters have been displayed. The command **./** (dot slash) re-displays the last variable.

You can use the special **sh** characters * (asterisk) and ? (question mark) in function and variable names, providing a limited form of pattern matching. If you give no function name, global variables and variables local to the current function are matched. If you specify a function name, then only variables local to that function are matched. To match only global variables, use the form *:pattern*.

The **sdb** command recognizes structures, arrays, and pointers so that all of the following commands work:

```
array[2][3]/
sym.id/
psym->usage/
xsym[20].p->usage/
```

*line*?[*lf*]
*variable*?[*lf*] Displays the value found in *objectfile* at the address selected by *line* or *variable* (function name), using the specified *l*ength and *f*ormat. The default format is **i**.

*line*=[*lf*]
*variable*=[*lf*]
*number*=[*lf*]

Displays the address of *variable* or *line* or the value of *number* in the specified length and format. The default is **lx**. *number*=[*lf*] provides a convenient way to convert decimal, octal, and hexadecimal values.

*variable*!*value*

Sets *variable* to the given *value*. *value* can be a numeric or character constant or another variable. Expressions that produce more than one value, such as structures, are not allowed as *value*. However, *variable* can be an expression which represents more than one variable, such as an array or structure name.

Specify a character constant with an initial ' (single quote), for example, **'c**. Numbers are treated as integers unless they contain a decimal point or an exponent. In the latter case, they are treated as having the type double. Register values are viewed as integers. If you give an address of a variable, it is treated as the address of a variable of type **int**. C conventions are used in any type conversions that are necessary to perform the indicated assignment.

**x**

Displays the machine registers and the current assembler language instruction.

**X**

Displays the current assembler language instruction.

## Displaying and Manipulating Source Files

**e** *function*
**e** *file*
**e** *dir*/
**e** *dir file*

Changes the current function, file, or directory. Specifying only *function* also sets the current file to the one containing the selected function. **sdb** reports the current function, file, or directory for any unspecified parameters.

*/pattern/*

Searches forward from the current line for a line containing a string matching *pattern*. The trailing / (slash) can be omitted. See "**ed**" on page 371 for a discussion of pattern notation.

*?pattern?*

Searches backward from the current line for a line containing a string matching *pattern*. The trailing **?** (question mark) can be omitted.

**p**

Displays the current line.

**z**

Displays the current line and the following nine lines. Sets the current line to the last line displayed.

**w**

Displays the 10 lines around the current line (a window).

*number*

Sets the current line to *number*. Displays the new current line.

*number*+ Advances the current line by the specified *number* of lines. Displays the new current line.

*number*- Decreases the current line by the specified *number* of lines. Displays the new current line.

**Ctrl-D** Scrolls. Pressing **Ctrl-D** displays the next 10 lines of source or data.

**Enter** If the previous command displayed a source line, pressing the **Enter** key advances the current line by one line and displays the new current line. If the previous command displayed a memory location, pressing the **Enter** key displays the next memory location.

## Controlling the Running of the Source Program

[*num*] **r** [*p* [*p2*] . . . ]
[*num*] **R** Runs the program with the given parameters. If you specify no parameters with **r**, it reuses previously specified parameters. **R** runs the program with no parameters. A parameter beginning with < (left angle bracket) or > (right angle bracket) redirects input or output, respectively. If given, *num* selects the number of breakpoints to be ignored.

[*line*] **b** [*command*[**;** *command*] . . . ]
Sets a breakpoint at the given line. If you specify a function name without a line number, **sdb** places a breakpoint at the first line in the function, even if it was not compiled with the **-g** flag. If you do not specify a *line*, a breakpoint is placed at the current line. If you specify no *command*s, the program stops running just before the breakpoint and returns control to **sdb**. Otherwise **sdb** performs the specified *command*s when the breakpoint is encountered, and then the program being debugged continues. If the **k** command is specified, however, control returns to **sdb**.

**B** Lists the currently active breakpoints.

[*line*] **d** Deletes a breakpoint at the selected line. If you select no *line*, breakpoints are deleted interactively. **sdb** displays each breakpoint location and reads a line from standard input. If the line begins with a **y** or **d**, then it deletes the breakpoint.

**D** Deletes all breakpoints.

[*line*] **c** [*num*]
[*line*] **C** [*num*] Continues running program after a breakpoint or an interrupt. **C** continues after resetting the signal that caused the program to stop. **c** ignores the signal. An optional *num* selects the number of breakpoints to ignore. If you specify a *line*, **sdb** places a temporary breakpoint at the line and continues the program. It deletes the breakpoint when the command finishes.

[*line*] **g** [*num*] Continues after a breakpoint, with execution resumed at the given line. *num* specifies how many breakpoints to ignore.

**l** Displays the last executed line.

**i**
**I** Runs the program one machine level instruction at a time, ignoring the signal that stopped the program (**i**) or passing the signal back to the program (**I**).

**s** [*num*]
**S** [*num*] Runs the program for one or the specified number of lines. **S** is equivalent to **s** except that it does not stop within called functions. Use **S** if you are confident that the called function works, but want to test the calling routine.

*variable***$m** [*num*]
*address*:**m** [*num*] Runs the program until the specified location is modified with a new value or is modified a specified *num* of times. The *variable* must be accessible from the current function.

*line* **a** If *line* is of the form *function*:*number*, this command has the effect of the **sdb** subcommand: *line* b 1. If *line* is of the form *function*:, it has the effect of the **sdb** subcommand: *function*: b 1.

[*level*] **v** Toggles verbose mode, for use with the **S, s** or **m** commands. If you omit *level*, then just the current source file or subroutine name is displayed when either changes. If *level* is 1 or greater, each C source line is displayed before it is executed; if *level* is 2 or greater, each assembler statement is also displayed. If verbose mode is on for any level, another **v** turns it off.

*function*(*p* [*p* . . . ])[/*f*]
Runs the named function, passing to it the specified parameters. These can be integer, character, or string constants or the names of variables accessible to the current function. You can specify the format of displayed values. The default format is **d** (decimal).

**k** Kills the program being debugged.

**M** Displays the address maps. Program addresses are mapped to file addresses using two field triples: *b1*, *e1*, *f1* and *b2*, *e2*, *f2*. The *f1* field is the length of the header at the beginning of the file; the *f2* field is the displacement from the beginning of the file to the data. For a plain executable file with mixed text and data, this is the same as the length of the header; for shared text and split instruction/data files, this is the length of the header plus the size of the text portion. The *b* and *e* fields are the starting and ending locations for a segment.

Given an address A, calculate its location in the file (either **a.out** or **core**) as follows:

If *b1* < *A* < *e1*
then *file address* = (*A*-*b1*) *f1*

If *b2* < *A* < *e2*
then *file address* = (*A*-*b2*) *f2*

**M**[**?**/][*****] *b e f* Records new values for the address map. The parameters **?** and / specify the text and data maps respectively. The first segment is changed unless you specify *****, in which case the second segment is changed. (These segments differ only in programs with split instruction and data space. In this case, use the second segment to examine the data section of the **a.out** file rather than the data section of the core image file.) If you give fewer than three values, the remaining map parameters are left unchanged.

*" string* Displays the given string. **sdb** recognizes C escape sequences of the form *\character*.

*!AIX-command* Performs the command specified by *AIX-command*.

*<file* Reads commands from *file* until reaching the **end-of-file** character and then continues to accept commands from standard input. When a command in such a file tells **sdb** to display a variable, the variable name is displayed along with the value. This command cannot be nested.

*>file* Redirects standard output to *file*.

**q** Exits the debugger.

### Debugging the Debugger

**Q** Displays a list of functions and files being debugged.
**V** Displays the version number.

## Files

a.out, core

## Related Information

The following commands: "**cc**" on page 140, "**ed**" on page 371, and "**sh**" on page 913.

The **a.out** and **core** files in *AIX Operating System Technical Reference*.

"Overview of International Character Support" in *Managing the AIX Operating System*.

"Debugging Programs" in *AIX Operating System Programming Tools and Interfaces*.

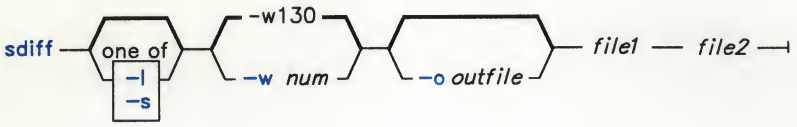
The discussion of Japanese Language Support in *Japanese Language Support User's Guide*.

# sdiff

## Purpose

Compares two files and displays the differences in a side by side format.

## Syntax




OL805301

## Description

The **sdiff** command reads *file1* and *file2*, uses **diff** to compare them, and writes the results to standard output in a side by side format. **sdiff** displays each line of the two files with a series of blanks between them if the lines are identical, a < (less than sign) in the field of blanks if the line only exists in *file1*, a > (greater than sign) if the line only exists in *file2*, and a | (vertical bar) for lines that are different.

When you specify the **-o** flag, **sdiff** produces a third file by merging *file1* and *file2* according to your instructions.

**Note:** The **sdiff** command invokes **diff -b** to compare two input files. The **-b** flag causes **diff** to ignore trailing spaces, tab characters, and consider other strings of blanks as equal.

## Flags

**-l** Displays only the left side when lines are identical.

**-o** *outfile* Creates a third file, *outfile*, by a controlled line-by-line merging of *file1* and *file2*. The following subcommands govern the creation of this file:

- **l** Adds the left side to *outfile*.
- **r** Adds the right side to *outfile*.
- **s** Stops displaying identical lines.

**v** Begins displaying identical lines.

**e l**
**e r**
**e b**
**e** Starts **ed** with the left side, the right side, both sides, or an empty file, respectively.

Each time you exit from **ed**, **sdiff** writes the resulting edited file to the end of *outfile*. If you fail to save the changes before exiting, **sdiff** writes the initial input to *outfile*.

**q** Exits the program.

**-s** Does not display identical lines.

**-w** *num* Sets the width of the output line to *num*, 130 characters, by default.

## Examples

1. To print a comparison of two files:

```
sdiff  chap1.bak  chap1  |  print
```

This prints a side by side listing that compares each line of `chap1.bak` and `chap1`. The `| print` sends the listing to the **print** command. **sdiff** assumes that your printer has wide paper (130 columns).

2. To display only the lines that differ:

```
sdiff  -s  -w 80  chap1.bak  chap1
```

This displays the differences at the work station. The **-w** 80 sets page width to 80 columns. The **-s** flag tells **sdiff** not to display lines that are identical in both files.

3. To selectively combine parts of two files:

```
sdiff -s  -w 80  -o chap1.combo  chap1.bak  chap1
```

This combines `chap1.bak` and `chap1` into a new file called `chap1.combo`. For each group of differing lines, **sdiff** asks you which group to keep or whether you want to edit them using **ed**.

## Related Information

The following commands: "**diff**" on page 320 and "**ed**" on page 371.

# secure

## Purpose

Establishes a more secure system configuration.

## Syntax


OL805480

## Description

The **secure** command increases the trustworthiness of the system. It configures trusted path management, login protection, and proper file system permissions. This command is a shell script and can be altered by an administrator of the system.

The **secure** command should be run after the initial system installation. This command can be run more than once, but it will not reverse any previous configuration set by the **secure** command.

### Trusted Path Management

For trusted path management, **secure** does the following:

- Adds the `sak = on` attribute to the default stanza of the **/etc/ports** file. This attribute enables the secure attention key (SAK) for the ports defined in **/etc/ports**.
- Adds the `synonym = /dev/hft` attribute to the **/dev/concole** stanza of the **/etc/ports** file. The addition of this attribute ensures that the protection and ownership of all the virtual terminals will match that of the console.
- Adds the `shell = /bin/actman` attribute to the **/dev/concole** stanza of the **/etc/ports** file. This addition will ensure that all the virtual terminals are closed before a user can log off the console.

### Login Protection

To enhance login protection, **secure** does the following:

- Removes any contents from **/etc/autolog**. This action prevents anyone from logging in without proper authentication.
- Prohibits users from logging in as superusers (**root**). Superuser privileges can still be obtained with the **su** command.
- Causes the default shell (or login shell) for **root** to be the trusted shell **/bin/tsh**.

### File System Permissions

For proper file system permissions, **secure** does the following:

- Checks the trusted computing base (TCB) to ensure proper permissions. In addition, **secure** identifies files not in the TCB that might be untrustworthy. This identification is done with the **sysck** command.
- Adds the **sysck** command to the **/etc/rc** file to run this command at system initialization.

---

**Japanese Language Support Information**

If Japanese Language Support is installed on your system, this command is not available.

---

## Related Information

The following command: "**sysck**" on page 1031.

The **sysck.cfg** and **ports** file formats in *AIX Operating System Technical Reference.*

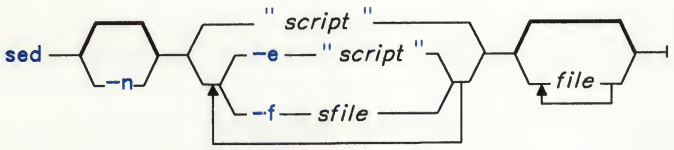
The discussion of security management in *Managing the AIX Operating System.*

# sed

## Purpose

Provides a stream editor.

## Syntax




OL805302

## Description

The **sed** command modifies lines from the specified *file* according to an edit script and writes them to standard output. The **sed** command includes many features for selecting lines to be modified and making changes only to the selected lines.

The **sed** command uses two work spaces for holding the line being modified: the ***pattern space***, where the selected line is held, and the ***hold space***, where a line can be stored temporarily.

An edit script consists of individual subcommands, each one on a separate line. The general form of **sed** subcommands is:

[*address-range*] *function*[*modifiers*]

The **sed** command processes each input *file* by reading an input line into a pattern space, applying all **sed** subcommands in sequence whose addresses select that line, and writing the pattern space to standard output. It then clears the pattern space and repeats this process for each line in the input *file*. Some of the subcommands use a hold space to save all or part of the pattern space for subsequent retrieval.

When a command includes an address, either a line number or a search pattern, only the addressed line or lines are affected by the command. Otherwise, the command is applied to all lines.

An address is either a decimal line number, a **$** (dollar sign), which addresses the last line of input, or a context address. A context address is a regular expression similar to those used in **ed** except for the following differences:

- You can select the character delimiter for patterns. The general form of the expression is:

  *\?pattern?*

  where *?* is a character delimiter you select. This delimiter cannot be a 2-byte international character support extended character. The default form for the pattern is:

  */pattern/*

- The sequence **\n** matches a new-line character in the pattern space, except the terminating new line.
- A . (dot) matches any character except a terminating new-line character. That is, unlike **ed**, which cannot match a new-line character in the middle of a line, **sed** can match a new-line character in the pattern space.

Certain commands allow you to specify one line or a range of lines to which the command should be applied. These commands are called ***addressed commands***. The following rules apply to addressed commands:

- A command line with no address selects every line.
- A command line with one address, expressed in context form, selects each line that matches the address.
- A command line with two addresses separated by commas selects the entire range from the first line that matches the first address through the next line that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

**Notes:**

1. The *text* parameter accompanying the **a**, **c**, and **i** commands can continue onto more than one line provided all lines but the last end with a \ to quote the new-line character. Back slashes in text are treated like back slashes in the replacement string of an **s** command and can be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* and *wfile* parameters must end the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins.
2. The **sed** command can process up to 99 commands in a pattern file.

## Flags

**-e** *"script"* Uses the string *script* as the editing script. If you are using just one **-e** flag and no **-f** flag, the **-e** flag can be omitted.

**-f** *sfile* Uses *sfile* as the source of the edit script. *sfile* is a prepared set of editing commands to be applied to *file*.

**-n** Suppresses all information normally written to standard output.

## Subcommands

In the following list of functions, the maximum number of permissible addresses for each function is indicated in parentheses. The **sed** script subcommands are as follows:

(1) **a**
*text* Places *text* on the output before reading the next input line.

(2)**b**[*label*] Branches to the **:** command bearing the *label*. If *label* is empty, it branches to the end of the script.

(2)**c**
*text* Deletes the pattern space. With 0 or 1 address or at the end of a 2-address range, places *text* on the output. Starts the next cycle.

(2)**d** Deletes the pattern space. Starts the next cycle.

(2)**D** Deletes the initial segment of the pattern space through the first new-line character. Starts the next cycle.

(2)**g** Replaces the contents of the pattern space by the contents of the hold space.

(2)**G** Appends the contents of the hold space to the pattern space.

(2)**h** Replaces the contents of the hold space by the contents of the pattern space.

(2)**H** Appends the contents of the pattern space to the hold space.

(1)**i**
*text* Writes *text* to standard output before reading the next line into the pattern space.

(2)**l** Writes the pattern space to standard output showing nondisplayable characters as 2- or 4-digit hexadecimal values. Long lines are folded.

(2)**n** Writes the pattern space to standard output. Replaces the pattern space with the next line of input.

(2)**N** Appends the next line of input to the pattern space with an embedded new-line character. (The current line number changes.) You can use this to search for patterns that are split onto two lines.

(2)**p** Writes the pattern space to standard output.

(2)**P** Writes the initial segment of the pattern space through the first new-line character to standard output.

(1)**q** Branches to the end of the script. Does not start a new cycle.

(2)**r** *rfile* Reads the contents of *rfile*. Places contents on the output before reading the next input line.

(2)**s**/*pattern*/*replacement*/*flags*
Substitutes *replacement* string for the first occurrence of the *pattern* in the pattern space. Any character appearing after the **s** can substitute for the / separator.

---

**Japanese Language Support Information**

Any single-byte character appearing after the **s** can substitute for the /.

---

You can add zero or more of the following *flags*:

**g** Substitutes all nonoverlapping instances of the *pattern* rather than just the first one.

**p** Writes the pattern space to standard out if a replacement was made.

**w** *wfile*
Writes the pattern space to *wfile* if a replacement was made. Appends the pattern space to *wfile*. If *wfile* was not already created by a previous write by this **sed** script, **sed** creates it.

(2)**t***label* Branches to **:***label* in the script file if any substitutions were made since the most recent reading of an input line execution of a **t** subcommand. If you do not specify *label*, control transfers to the end of the script.

(2)**w***wfile* Appends the pattern space to *wfile*.

(2)**x** Exchanges the contents of the pattern space and the hold space.

(2)**y**/*pattern1*/*pattern2*/
Replaces all occurrences of characters in *pattern1* with the corresponding characters *pattern2*. The byte lengths of *pattern1* and *pattern2* must be equal.

---

**Japanese Language Support Information**

The character lengths of these two patterns must be equal.

---

(2)!*sed-cmd* Applies the specified **sed** subcommand only to lines *not* selected by the address or addresses.

(0):*label* This script entry simply marks a branch point to be referenced by the **b** and **t** commands. This label can be any sequence of eight or fewer bytes.

(1)= Writes the current line number to standard output as a line.

(2){*subcmd*

.
.
.

} Groups subcommands enclosed in {} (braces).

## Examples

1. To perform a global change:

```
sed  "s/happy/enchanted/g"  chap1  >chap1.new
```

This replaces each occurrence of `happy` found in the file `chap1` with `enchanted`, and puts the edited version in a separate file named `chap1.new`. The `g` at the end of the **s** subcommand tells **sed** to make as many substitutions as possible on each line. Without the `g`, **sed** replaces only the first `happy` on a line.

The **sed** stream editor operates as a filter. It reads text from standard input or from the files named on the command line (`chap1` in this example), modifies this text, and writes it to standard output. Unlike most editors, it does not replace the original file. This makes **sed** a powerful command when used in pipelines.

2. To use **sed** as a filter in a pipeline:

```
pr  chap2 ¦ sed  "s/Page  *[0-9]*$/(&)/" ¦ print
```

This encloses the page numbers in parentheses before printing `chap2`. The **pr** command puts a heading and page number at the top of each page, then **sed** puts the page numbers in parentheses, and the **print** command prints the edited listing.

The **sed** pattern `/Page *[0-9]*$/` matches page numbers that appear at the end of a line. The **s** subcommand changes this to `(&)`, where the & stands for the page number that was matched.

3. To display selected lines of a file:

```
sed  -n  "/food/p" chap3
```

This displays each line in `chap3` that contains the word `food`. Normally, **sed** copies every line to standard output after it is edited. The **-n** flag stops **sed** from doing this. You then use subcommands like **p** to write specific parts of the text. Without the **-n**,

this example would display *all* the lines in `chap3`, and it would show each line containing `food` twice.

4. To perform complex editing:

```
sed  -f  script.sed  chap4  >chap4.new
```

It is always a good idea to create a **sed** script file when you want to do anything very complex. You can then test and modify your script before using it. You can also reuse your script to edit other files. Create the script file with an interactive text editor.

5. A sample **sed** script file:

```
:join
/\\$/{N
s/\\\n//
b join
}
```

This **sed** script joins each line that ends with a \ (backslash) to the line that follows it. First, the pattern `/\\$/` selects a line that ends with a \ for the group of commands enclosed in `{ }`. The **N** subcommand then appends the next line, imbedding a new-line character. The `s/\\\n//` deletes the \ and imbedded new-line character. Finally, **b** `join` branches back to the label `:join` to check for a \ at the end of the newly joined line. Without the branch, **sed** writes the joined line and read the next one before checking for a second \.

**Note:** The **N** subcommand causes **sed** to stop immediately if there are no more lines of input (that is, if **N** reads an end-of-file character). It does not copy the pattern space to standard output before stopping. This means that if the last line of the input ends with a \, then it is not copied to the output.

## Related Information

The following commands: "**awk**" on page 81, "**ed**" on page 371, and "**grep**" on page 501.

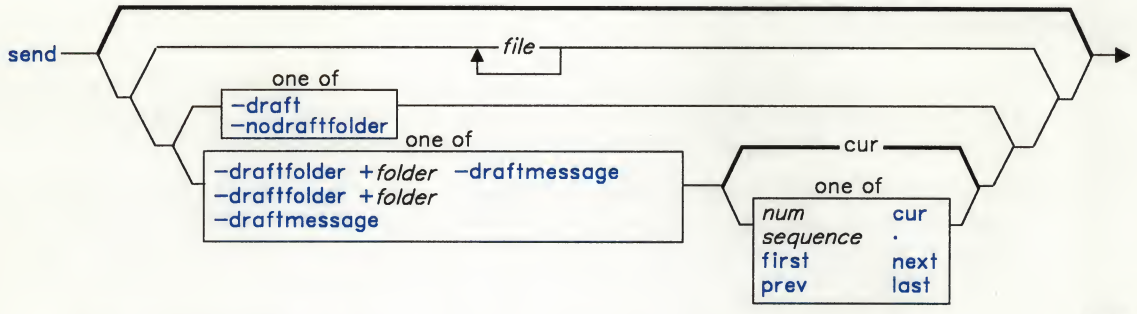
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

# send

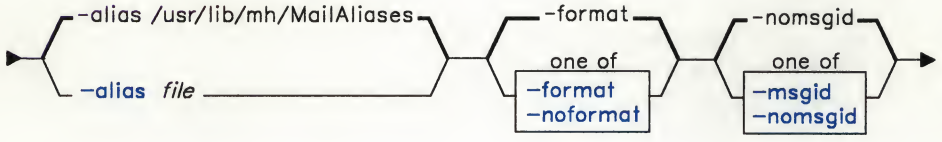
## Purpose

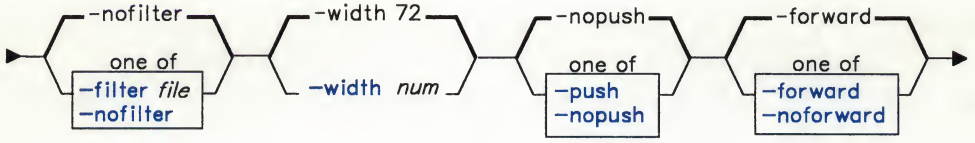
Sends a message.

## Syntax

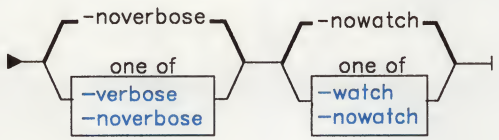


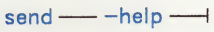

AJ2FL245




AJ2FL246




AJ2FL170

## Description

The **send** command is used to route messages to the mail delivery system by way of the **post** command or the **spost** command. **send** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **send** command causes **From:** and **Date:** fields to be added to each specified message. **send** places the sender's address in the **From:** field unless a **$SIGNATURE** environment variable is set or a **signature:** profile entry is present. In either of these cases **send** uses the signature. **send** puts the current date in the **Date:** field. If the **dist** command calls **send**, **send** prepends **Resent-** to the **From:**, **Date:**, and **Message-ID:** fields.

The **send** command then takes each of the specified message files and calls the **post** command or the **spost** command to deliver them. After successful delivery, **send** renames the message file by placing a comma in front of the file name. This comma removes the message from the folder without actually deleting the file. The file is retrievable until you send another message. If the delivery fails, **send** displays an error message.

## Flags

| | |
|---|---|
| **-alias** *file* | Specifies that *file* is a mail alias file to be searched for aliases. The default alias file is **/usr/lib/mh/MailAliases**. |
| **-draft** | Uses the current draft message if no file is specified. Without this flag, **send** asks the user if the current draft message is the one to use when no file is specified. |
| **-draftfolder** +*folder* | Specifies the draft folder that contains the draft message to be sent. If **-draftfolder** +*folder* is followed by *msg*, *msg* represents the **-draftmessage** attribute. |

**-draftmessage** *msg* Specifies the draft message to be sent. You can use one of the following message references as *msg*:

| | | |
|---|---|---|
| *num* | *sequence* | **first** |
| **prev** | **cur** | **.** |
| **next** | **last** | **new** |

The default draft message is **cur**.

**-filter** *file* Uses the format instructions in the specified file to reformat copies of the message sent to **Bcc:** recipients.

**-format** Puts all recipient addresses in a standard format for the delivery transport system. This flag is the default.

**-forward** Adds a failure message to the draft message and returns it to the sender if the **send** command fails to deliver the draft. This flag is the default.

**-help** Displays help information for the command.

**-msgid** Adds a message-identification component (such as **Message-ID:**) to the message.

**-nodraftfolder** Undoes the last occurrence of **-draftfolder** +*folder*. This flag is the default.

**-nofilter** Removes the **Bcc:** header from the message for recipients listed in the **To:** and **cc:** fields, and sends the message with minimal headers to recipients listed in the **Bcc:** field. This flag is the default.

**-noformat** Does not alter the format of the recipient addresses.

**-noforward** Does not return the draft message to the sender if delivery fails.

**-nomsgid** Does not add a message-identification component. This flag is the default.

**-nopush** Runs the **send** command in the foreground (see the **-push** flag). This flag is the default.

**-noverbose** Does not display information during the delivery of the message to the **sendmail** command. This flag is the default.

**-nowatch** Does not display information during delivery by the **sendmail** command. This flag is the default.

**-push** Runs the **send** command in the background. **send** does not display error messages on the terminal if delivery fails. You can use the **-forward** flag to return messages to you that fail on delivery.

| | |
|---|---|
| **-verbose** | Displays information during the delivery of the message to the **sendmail** command. This information allows you to monitor the steps involved. |
| **-watch** | Displays information during the delivery of the message by the **sendmail** command. This information allows you to monitor the steps involved. |
| **-width** *num* | Sets the width of components that contain addresses. The default is 72 columns. |

## Profile Entries

| | |
|---|---|
| **Draft-Folder:** | Sets your default folder for drafts. |
| **mailproc:** | Specifies the program used to post failure notices. |
| **Path:** | Specifies your *user_mh_directory*. |
| **postproc:** | Specifies the program used to post messages. |
| **Signature:** | Sets your mail signature. |

## Files

| | |
|---|---|
| $HOME/.mh_profile | The MH user profile. |

## Related Information

The following commands: "**ali**" on page 48, "**comp**" on page 185, "**dist**" on page 336, "**forw**" on page 438, "**post**" on page 758, and "**sendmail**" on page 897.

The **mh-alias**, **mh-format**, and **mh-profile** files in *AIX Operating System Technical Reference*.

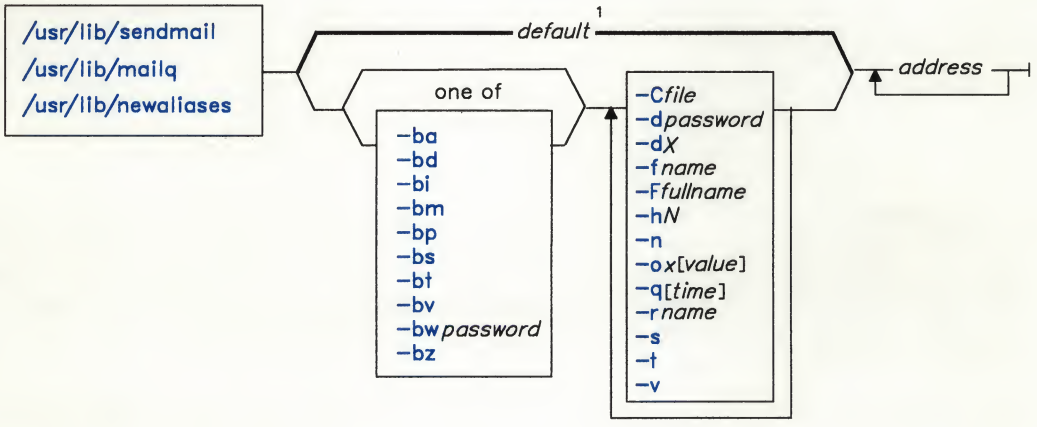
"Overview of the Message Handling Package" in *Managing the AIX Operating System*.

# sendmail

## Purpose

Routes mail for local or network delivery.

## Syntax




[1]*default* = -bm for **sendmail**
-bp for **mailq**
-bi for **newaliases**

AJ2FL126

/usr/lib/mailq —|

AJ2FL145

/usr/lib/newaliases —|

AJ2FL147

## Description

The **sendmail** program receives formatted text messages and routes the message to one or more other users on the local system, or if connected to a network, to users on other systems. The program translates the format of message heading information to match the requirements of the destination system. It determines which network to use based on the syntax and content of the addresses.

The program can deliver messages to:

- Users on the local system
- Users connected to the local system using the TCP/IP protocol
- Users connected to the local system using the **uucp** protocol.

The **sendmail** program operates mainly as a background, mail-routing program. Other programs, such as **Mail** and the message handler routines, provide user interfaces for generating and receiving mail that **sendmail** handles. However, if you enter the **sendmail** command with no flags, it reads standard input for the message text until it receives a **Ctrl-D** or a line with only a single period, designating the end of the message. Then it sends a copy of the message to all addresses listed. For example, the following input at the command line sends the message `This is a test message` to the mailbox for user `george` on the local system:

```
$ /usr/lib/sendmail george
This is a test message
.
$
```

The **sendmail** program uses a configuration file (**/usr/adm/sendmail/sendmail.cf** by default) to set many operational parameters and to determine how the program parses addresses. This file is a text file that you can edit. However, **sendmail** uses a compiled form of this file (**/usr/adm/sendmail/sendmail.cfDB**). For any changes made to this file to be effective, you must build a new copy of the compiled configuration file by running **sendmail** with the **-bz** flag.

The **sendmail** program also allows you to define aliases to use when addressing mail handled by the local **sendmail** program. ***Aliases*** are alternate names that can be used in place of elaborate network addresses. You can also use aliases to build distribution lists. Define aliases in **/usr/adm/sendmail/aliases**. This file is a text file you can edit. However, **sendmail** uses a database version of this file that is kept in the directory **/usr/adm/sendmail/aliasesDB**. For any changes made to the aliases file to be effective, you must build a new alias database by running **sendmail** with the **-bi** flag. If the **sendmail** daemon is running, you must also stop that process and start the daemon again before it recognizes the new alias data base file. Normally the sender of a message is not included when **sendmail** expands an alias address. For example, if `amy` sends a message to alias `D998` and she is defined as a member of that alias, **sendmail** does not send a copy of the message to `amy`.

Every system must have a user or user alias designated ***postmaster***. Assign this alias in the file **/usr/adm/sendmail/aliases**. Unless you change the entry in this file, this alias is assigned to `root`. This alias allows other users outside your system to send mail to a known ID (i.e. `postmaster`) to get information about mailing to users on your system. Also, users on your system can send problem notifications to this ID.

Two additional commands are links to **sendmail**:

| | |
|---|---|
| **/usr/lib/mailq** | Prints the contents of the mail queue. This command is the same as running **sendmail** with the **-bp** flag. |
| **/usr/lib/newaliases** | Builds a new copy of the alias data base from the file **/usr/adm/sendmail/aliases**. This command is the same as running **sendmail** with the **-bi** flag. |

## Mail Addresses

Mail addresses are based on the domain address (Arpanet) protocol. These addresses have the following form:

*user@host.domain*

**Note:** The configuration file provided with **sendmail** specifies that blanks in addresses be converted to periods before being transmitted. This convention follows the Arpanet mail protocol as described in RFC822, but does not match the Arpanet mail protocol as described in RFC733 (NIC41952). You can change this setting by setting the **OB** option in the **sendmail** configuration file.

A ***domain*** is a logical grouping of systems that are connected together by physical network links. No direct relationship exists between the actual physical interconnections and the way in which the systems are grouped in the domain. The ***domain name*** identifies a specific domain within a larger group of domains. The domain name has the format of a tree structure. Each node (or leaf) on the tree corresponds to a resource set, and each node can create and contain new domains below it. The actual domain name of a node is the path from the root of the tree to that node. Domain names do not correspond to system names, host addresses, or any other type of information.

For example, if node `hera` is part of the domain `IBM`, which is in turn a subdomain of `COM`, and a message is sent to `geo` at that address it is sent to:

```
geo@hera.IBM.COM
```

The message router (usually **sendmail**) must determine how to send the message to its final destination. If the router is at `hera`, it delivers the message to user `geo`. If the router is at another system within the `IBM` domain, it corresponds with the name server for that domain to find out how to deliver the message. If the router is not a part of the `IBM` domain, but is in a domain that is under the `COM` domain, it corresponds with the name server for the `COM` domain to find out how to deliver the message. The respective name server returns a network address to the router. That network address determines the actual path that the message takes to its destination.

The domain address is read from right to left, with each domain in the address separated from the next domain with a . (period). This format does not imply any routing. Thus, although the example is specified as a COM address, the message might actually travel by a different route if that were more convenient or efficient. At some sites, the message associated with the sample address might go directly from the sender to node hera over a local area network. At other sites it might be sent over a **uucp** network or a combination of other delivery methods.

Normally, the actual routing of a message is handled automatically. However, you can route the message manually through several specified hosts to get it to its final destination. An address using intermediate hosts, called a ***route address***, has the following form:

*@hosta,@hostb:user@hostc*

This address specifies that the message should go first to the remote system represented by *hosta*, then to the remote system represented by *hostb*, and finally to the remote system represented by *hostc*. This path is forced even if there is a more efficient route to *hostc*.

In some cases the user can abbreviate the address rather than typing the entire domain name. In general, systems in the same domain do not need to use the full domain name. For example, a user on node zeus.IBM.COM can send a message to geo@hera.IBM.COM by typing only geo@hera, because they are in the same local domain, IBM.COM.

Other mail address formats exist that are used by other mail routing programs (for example, **uucp**). The mail routing program (**sendmail**) converts most of these other formats to a format that the network routing system can use. However, if you use the domain address format, the routing program operates more efficiently.

For example, if **sendmail** receives an address in the following format:

*@host:user*

It converts it to the corresponding domain address format:

*user@host*

Similarly, if **sendmail** receives an address in the following format:

*host!user*

The mail routing program routes the message directly to the **uucp** command (part of the Basic Networking Utilities (BNU)) However, when sending mail via **uucp**, you must include a route address that indicates which BNU host(s) to send the message through to get to the final destination.

To route messages through the BNU network, use one of the following domain address formats. Your choice depends on the way in which the systems at your site are connected:

1. *@system_name.domain_name:uucp-route!user-ID*

   For example, the address:

   ```
   @zeus:hera!amy
   ```

   sends a message to user amy on **uucp** host hera by way of system zeus. The address:

   ```
   @apollo.802:merlin!lgh
   ```

   sends a message to user lgh on **uucp** host merlin via system apollo under the local domain 802.

2. *uucp-route!user-ID@system_name.domain_name*

   In this case, the address:

   ```
   merlin!arthur!amy@hera.802
   ```

   sends a message to user amy on system hera under domain 802 via the BNU link merlin through arthur.

3. *system_name.domain_name:uucp-route!user-ID@system_ name.domain_name*

   In this example, the address:

   ```
   @apollo.802:merlin!arthur!amy@hera.802
   ```

   sends a message to user amy on system hera under domain 802 that first goes through apollo, the gateway node for domain 802, and then through the BNU link merlin through arthur. (Including 802 in this example is optional, since the two domain names are identical.)

4. *hosta!hostb!hostc!user*

   This example is a purely **uucp** route address:

   ```
   zeus!hera!kronos!amy
   ```

   sends a message to amy on kronos, via the BNU link zeus through hera.

5. *@hosta*.UUCP:*@hostb*.UUCP:*user@hostc*

   This example, like the previous one, is a purely **uucp** route address:

   ```
   @zeus.UUCP:@hera.UUCP:amy@kronos.UUCP
   ```

   sends a message to amy on kronos, via the BNU link zeus through hera.

## Return Codes

The **sendmail** program returns an exit status code. These exit codes are defined in **/usr/include/bsd/sysexits.h**. The following table summarizes the meanings of these return codes:

| Return Code | Meaning |
|---|---|
| **EX_CANTCREAT** | The **sendmail** program cannot create a file that the user specified. |
| **EX_DATAERR** | The user's input data was not correct. |
| **EX_IOERR** | An error occurred during I/O. |
| **EX_NOHOST** | The **sendmail** program could not recognize the specified host name. |
| **EX_NOINPUT** | The **sendmail** program either could not find, or could not read, the specified input file. |
| **EX_NOPERM** | The user does not have the needed permissions to perform the requested operation. |
| **EX_NOUSER** | The **sendmail** program could not recognize a specified user ID. |
| **EX_OK** | The **sendmail** program successfully completed the operation for all addresses. |
| **EX_OSERR** | A temporary operating system error occurred. An example of such an error is `cannot fork because too many processes are currently running`. |
| **EX_OSFILE** | A system file error occurred. For example, a system file (such as **/etc/passwd**) does not exist, cannot be opened or has another type of error preventing it from being used. |
| **EX_PROTOCOL** | The remote system returned something that was incorrect during a protocol exchange. |
| **EX_SOFTWARE** | An internal software error occurred (including bad arguments). |

| Return Code | Meaning |
|---|---|
| **EX_TEMPFAIL** | The **sendmail** program could not create a connection. Try the request again later. |
| **EX_UNAVAILABLE** | A service or resource that **sendmail** needed was not available. |
| **EX_USAGE** | The command syntax was not correct. |

## Flags

| | |
|---|---|
| **-ba** | Starts **sendmail** in Arpanet mode. All input lines to the program must end with a carriage return and a line feed (**CR-LF**). The program generates messages with a **CR-LF** at the end. The program looks at the `from` and the `sender` fields to find the name of the sender. |
| **-bd** | Starts **sendmail** to run in the background as a TCP/IP mail routing daemon. |
| **-bi** | Builds the alias database files from information defined in **/usr/adm/sendmail/aliases**. Running **sendmail** with this flag is the same as running the command, **/usr/lib/newaliases**. |
| **-bm** | Delivers mail in the usual way. This is the default. |
| **-bp** | Prints a listing of the mail queue. Running **sendmail** with this flag is the same as running the command, **/usr/lib/mailq**. |
| **-bs** | Uses the simple mail transfer protocol (SMTP) as described in RFC821. This flag implies all of the operations of the **-ba** flag that are compatible with SMTP. |
| **-bt** | Starts **sendmail** in address test mode. This mode allows you to enter addresses interactively and watch as **sendmail** displays the steps it takes to parse the address. Use this mode for debugging the address parsing rules in a new configuration file. |
| **-bv** | Starts **sendmail** with a request to verify the user IDs provided in the *address* field of the command. The program responds with a message telling which IDs can be resolved to a mailer program. It does not try to collect or deliver a message. Use this mode to validate the format of user IDs, aliases, or mailing lists. |
| **-bz** | Builds the compiled version of the configuration file from information in **/usr/adm/sendmail/sendmail.cf**. |
| **-C***file* | Starts **sendmail** using an alternate configuration file specified by the *file* parameter. Use this flag together with **-bt** to test a new configuration file before installing it as the running configuration file. |

| | |
|---|---|
| **-d***X* | Sets debugging value to *X*. |
| **-F***fullname* | Sets the full name of the sender to be the string provided in the *fullname* parameter. |
| **-f***name* | Sets the name of the *from* person (the sender of the mail). This flag can be used only by those administrative user IDs designated in the configuration file with the **T** control line, or if your ID is the ID supplied in *name*. |
| **-h***N* | Sets the hop count to *N*. The ***hop count*** is the number of times that the message has been processed by a **sendmail** program (not just the local copy of **sendmail**). The program increments the hop count every time the message is processed. When it reaches a limit, the message is returned with an error message usually caused by alias looping. |
| **-n** | Does not do aliasing. |
| **-o***x*[*value*] | Sets option *x*. If the option is a valued option, you must also specify *value*. See Figure 10 on page 905 for possible options, values, and their meanings. |
| **-q**[*time*] | Processes saved messages in the queue at the intervals specified by *time*. If *time* is not specified, this flag processes the queue at once. You can specify *time* as a tagged number using the tag *s* for seconds, *m* for minutes, *h* for hours, *d* for days, and *w* for weeks. If the tag letter is omitted and just a number is specified, **sendmail** uses days as the unit of time. For example, `-q2m` processes the queue every two minutes, but `-q2` processes the queue every two days. |
| **-r***name* | An alternate and obsolete form of the **-f** flag. |
| **-t** | Reads the `To:`, `Cc:`, and `Bcc:` lines of the message header to determine where to send the message; deletes the `Bcc:` line before transmitting the message; and deletes any addresses in the argument list of the **sendmail** command. |
| **-v** | Starts **sendmail** in verbose mode. The program displays messages regarding the status of transmission, the expansion of aliases and any errors that may occur during the sending of the message. |

You can also set or remove **sendmail** processing options. Normally, the person responsible for the mail system uses these options. To set these options, use the **-o** flag on the command line or the `O` control line in the configuration file (**sendmail.cf**).

| Option | Function |
|---|---|
| A*file* | Uses the named *file* as an alternate alias file. |
| B*c* | Sets the blank substitution character to the character specified in the parameter *c*. The **sendmail** program replaces unquoted spaces in addresses with this character. The supplied configuration file uses the period (.) for this character. |
| **c** | If an outgoing mailer program is specified in the configuration file as expensive to use, this option causes **sendmail** to queue messages for that mailer program without sending them. The queue can be run later when costs are lowest or when the queue is large enough to send the messages efficiently. |
| d*x* | Sets the delivery mode to *x*. Delivery modes are **i** for interactive (synchronous) delivery, **b** for background (asynchronous) delivery, and **q** for queue only (next time the queue is run) delivery. The supplied configuration file uses a value of **b**. |
| e*x* | Sets error processing to mode *x*. Valid modes are:<br>**m** Mails the error message to the user's mailbox.<br>**w** Writes the error message to the terminal or mails it if the user is not logged in.<br>**p** Displays the error message on the terminal (default).<br>**q** Throws away error message and returns the exit status only.<br>**e** Mails the error message to the user's mailbox, but always exits with a zero exit status (normal return).<br>If the text of the message is not mailed by modes **m** or **w** and if the sender is a local user, a copy of the message is appended to the file **dead.letter** in the sender's home directory. |
| **f** | Saves `From` lines at the front of messages. These lines are normally discarded. |
| g*N* | Sets the default group ID to use when calling mailers to the value specified by *N*. |
| H*file* | Specifies the name of the SMTP help file (**/usr/adm/sendmail/sendmail.hf** by default). |

**Figure 10 (Part 1 of 3). Configuration Options**

| Option | Function |
|---|---|
| **i** | Does not interpret a period (.) on a line by itself as a message terminator. |
| **I***c* | Interpret the special character specified by *c* as a space character and all real spaces as delimiters when processing address parsing rules. Use this control line if using an editor that saves tabs as spaces (such as **INed**) to edit the configuration file. The default setting of this line is I_. |
| **L***n* | Specifies the log level to be the value supplied in the *n* parameter. Valid levels and the activities that they represent are (each number includes the activities of all numbers of lesser value and adds the activity that it represents): |

| Level | Activity Logged |
|---|---|
| **0** | Prevents logging. |
| **1** | Logs major problems only. |
| **2** | Logs message collections and failed deliveries. |
| **3** | Logs successful deliveries. |
| **4** | Logs messages deferred (for example, because the host is down). |
| **5** | Logs messages that are placed in the queue (normal event). |
| **6** | Logs unusual but benign incidents (for example, trying to process a locked file). |
| **9** | Logs internal queue ID to external message ID mappings. This can be useful for tracing a message as it travels between several hosts. |
| **12** | Logs messages that are of interest when debugging. |
| **16** | Logs verbose information regarding the queue. |

| | |
|---|---|
| **M***x value* | Sets the macro *x* to *value*. Use this option from the command line only (with the **-o** flag). |
| **m** | Sends to the sender (me) also, if the sender is in an alias expansion. Normally, the sender does not receive a copy of the message. |
| **N***netname* | Sets the name of the host network to *netname*. The **sendmail** program compares the argument of an SMTP **HELO** command to *hostname.netname* (it gets the value of *hostname* from the kernel). If these values do not match, it adds the *hostname.netname* string to the `Received:` lines in the message so that messages can be traced accurately. |

**Figure 10 (Part 2 of 3). Configuration Options**

| Option | Function |
|---|---|
| **o** | This option indicates that this message may have old style headers. Without this option, the message has new style headers (commas instead of spaces between addresses). If this option is set, an adaptive algorithm correctly determines the header format in most cases. |
| **Q***dir* | Sets the directory in which to queue messages to the directory specified by the *dir* parameter. That directory must exist. |
| **r***time* | Sets the timeout for reads from a mailer program to the value specified by *time*. If no timeout value is set, **sendmail** waits indefinitely for a mailer to respond. The default value for this timeout is 5 minutes. |
| **S***file* | Sets the mail statistics file to the *file*. If this file exists, **sendmail** stores statistics about mail traffic in a database format in this file. Use the **mailstats** command to read the information in this file. If the indicated file does not exist, no statistic information is saved. |
| **s** | Interactive mode delivers mail without going through the mail queue. When this option is specified, mail is passed through the mail queue in interactive mode also. This action ensures that the message being sent is not lost if a delivery problem occurs. |
| **T***time* | Sets the timeout on messages in the queue to the specified *time*. After a message has been in the queue for this amount of time, **sendmail** returns the message to the sender. In **sendmail.cf** that is provided with **sendmail**, this value is set to three days. |
| **u***N* | Sets the default user ID to use when calling mailers to the value specified by *N*. |
| **v** | Run in verbose mode. |
| **Y** | When this option is specified, **sendmail** delivers each message in the mail queue from a separate process. This option uses less memory to process the mail queue. Use of this option is not recommended. |

**Figure 10 (Part 3 of 3). Configuration Options**

## Files

| | |
|---|---|
| /usr/lib/sendmail | Contains the **sendmail** program. |
| /usr/lib/mailq | Displays list of the mail queue. |
| /usr/lib/newaliases | Builds alias database. |

| | |
|---|---|
| /usr/lib/mailstats | Displays **sendmail** statistics found in **/usr/adm/sendmail/sendmail.st**. |
| /usr/lib/sendmail.sh | Contains a shell script replacement for **sendmail** when **sendmail** is not installed. Only local mail can be delivered with this shell script acting as **sendmail**. |
| /usr/adm/sendmail/aliases | Contains the text version of **sendmail** aliases. |
| /usr/adm/sendmail/aliasesDB/DB.dir | Contains one of the alias database files. |
| /usr/adm/sendmail/aliasesDB/DB.pg | Contains one of the alias database files. |
| /usr/adm/sendmail/aliasesDBl | Contains the alias database lock file. |
| /usr/adm/sendmail/sendmail.hf | Contains the SMTP help file. |
| /usr/adm/sendmail/sendmail.cf | Contains the text version of the **sendmail** configuration file. |
| /usr/adm/sendmail/sendmail.cfDB | Contains the **sendmail** configuration database file. |
| /usr/adm/sendmail/sendmail.cfDBl | Contains the **sendmail** configuration database lock file. |
| /usr/adm/sendmail/sendmail.st | Contains the **sendmail** statistics file. |
| /usr/adm/sendmail/smdemon.cleanu | Maintains aging copies of the **log** file in **/usr/spool/mqueue**. |
| /etc/rc.sendmail | Contains the shell script to start the **sendmail** daemon. |
| /usr/spool/mqueue | Contains the **log** file and temporary files associated with the messages in the mail queue (the mail queue directory). Temporary files have names that include the mail queue ID (*mqid*) of the message for which the file was created:<br>df*mqid* Data file<br>lf*mqid* Lock file<br>nf*mqid* Backup file<br>qf*mqid* Queue control file<br>tf*mqid* Temporary control file<br>xf*mqid* Transcript file for session. |
| /usr/spool/cron/crontabs/root | Contains a commented entry to run **sendmail** periodically for use when not routing mail to a network. Uncomment that entry to process the mail queue at the interval specified in that **cron** file. |
| /bin/uux | Contains the mailer program to deliver **uucp** mail. |
| /bin/bellmail | Contains the mailer program to deliver local mail. |

## Related Information

The following commands: "**bellmail**" on page 104, "**mail, Mail**" on page 608, "**uux**" on page 1166.

The book *Interface Program for use with TCP/IP.*

The chapter about managing the mail system in *IBM RT Managing the AIX Operating System.*

The chapter about sending and receiving mail in *IBM RT Using the AIX Operating System.*

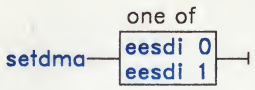
The file **sendmail.cf** in *AIX Operating System Technical Reference.*

# setdma

## Purpose

Sets the DMA channel of the specified adapter.

## Syntax




OL805466

## Description

The **setdma** command sets the DMA channel of the specified adapter. The **eesdi 0** sets the DMA adapter channel to 0. The **eesdi 1** sets the DMA adapter channel to 1.

## Related Information

*Installing and Customizing the AIX Operating System.*

# setmnt

## Purpose

Creates mount table.

## Syntax


OL805062

## Description

The **setmnt** command reads lines from standard input and writes the **/etc/mnttab** table to standard output (see the **mnttab** file in *AIX Operating System Technical Reference*). The **/etc/mmttab** is needed for both the **mount** and **unmount** commands. **setmnt** creates a **mnttab** entry for each line read. Input lines have the format:

*filesys directory*

where *filesys* is the name of the file system's special file and *directory* is the root name of that file system. Thus, *filesys* and *directory* become the first two strings in the **mnttab** entry. *filesys* and *directory* must not be longer than 100 characters.

The **setmnt** command enforces an upper limit on the maximum number of **mnttab** entries.

## Example

To set the mount table after it has been destroyed or made invalid:

```
setmnt  <<end
hd1  /u
fd0  /a
fd1  /b
end
```

This command sets the mount table to show **/dev/hd1** mounted on /u, **/dev/fd0** on /a, and **/dev/fd1** on /b.

The <<end and end define a Here Document, which uses the text entered before the end line as the standard input for the **setmnt** command. For more details, see "Inline Input Documents" on page 928.

## Files

/etc/mnttab

## Related Information

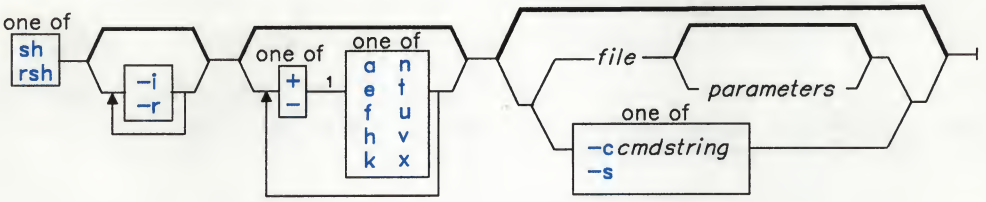
The **mnttab** file in *AIX Operating System Technical Reference.*

# sh

## Purpose

Interprets commands read from a file or entered at the keyboard.

## Syntax




OL805425

[1] Do not put a blank between these items.

OL805308

## Description

The **sh** command is a system command interpreter and programming language. It is not a part of the operating system ***kernel***, but an ordinary user program that reads commands entered at the keyboard and arranges for their execution. In addition, it can read commands that you have saved in a file. Such a file is usually called a ***shell procedure*** or a ***command file***. For a complete description of how to write shell procedures to take advantage of this useful tool, see *Using the AIX Operating System*.

A restricted version of shell (the **Rsh** command) is available in the **/bin/Rsh** file that allows you to create user environments with a limited set of privileges and capabilities. See "Restricted Shell" on page 935 for additional information on the restricted shell.

### Commands

A ***command*** is either a simple command or a ***control command*** (see "Control Commands" on page 930).

A ***simple command*** is a sequence of ***words*** separated by blanks or tabs. A word is a sequence of characters and/or numerals that contains no unquoted blanks. The first word in the sequence (numbered as 0), usually specifies the name of a command. Any remaining words, with a few exceptions, are passed to that command.

The ***value*** of a simple command is its exit value if it ends normally or (octal) 200 *status* if it ends abnormally. For a list of *status* values, see the **signal** system call in *AIX Operating System Technical Reference.*

A ***pipeline*** is a sequence of one or more commands separated by a | (vertical bar) or, for historical compatibility, by a ^ (circumflex). In a pipeline, the standard output of each command becomes the standard input of the next command. Each command runs as a separate process, and the shell waits for the last command to end. A ***filter*** is a command that reads its standard input, transforms it in some way, then writes it to its standard output. A pipeline normally consists of a series of filters. Although the processes in a pipeline (except the first process) can execute in parallel, they are synchronized to the extent that each program needs to read the output of its predecessor.

The exit value of a pipeline is the exit value of the last command.

A ***list*** is a sequence of one or more pipelines separated by ; (semicolon), & (ampersand), && (two ampersands), or || (two vertical bars) and optionally ended by a ; (semicolon) or an & (ampersand). These separators and terminators have the following effects:

; Causes ***sequential execution*** of the preceding pipeline (the shell waits for the pipeline to finish).

& Causes ***asynchronous execution*** of the preceding pipeline the shell does *not* wait for the pipeline to finish).

&& Causes the list following it to be executed *only* if the preceding pipeline returns a zero exit value.

|| Causes the list following it to be executed *only* if the preceding pipeline returns a nonzero exit value.

**Note:** The **cd** command is an exception. If it returns a nonzero exit value, no subsequent commands in a list are executed, regardless of the separator characters.

The ; and & separators have equal precedence, as do && and ||. The single-character separators have lower precedence than the double-character separators. An unquoted new-line character following a pipeline functions the same as a ; (semicolon).

The shell treats as a comment any word that begins with a # character and ignores that word and all characters following up to the next new-line character.

## Command Execution

Each time the shell executes a command, it carries out the substitutions discussed in the following text. If the command name matches one of the built-in commands discussed in "Built-in Commands" on page 931, it executes it in the shell process. If the command name does not match a built-in command but matches the name of a defined function, it executes the function in the shell process. The shell sets the positional parameters to the parameters of the function.

If the command name matches neither a built-in command nor the name of a defined function and the command names an executable file that is a compiled (binary) program, the shell (as ***parent***) spawns a new (***child***) process that immediately runs the program. If the file is marked executable but is not a compiled program, the shell assumes that it is a shell procedure. In this case, the shell spawns another instance of itself (a ***subshell***), to read the file and execute the commands included in it (note how this differs from the execution of functions). The shell also executes a parenthesized command in a subshell (see page 931). From your point of view as a user, a compiled program is run in exactly the same way as a shell procedure.

The shell normally searches for commands in four places in the file system. The shell first looks for the command in the **/bin** directory. If it does not find the command there, it looks in the **/usr/bin** directory. If this also fails, it looks in the **/etc** directory and then, finally, in the current directory. You can also give a specific path name when you invoke a command, for example `/bin/sort`, in which case the shell does not search any directories other than the one you specify in the path name. If the command name contains a / (slash), the shell does not use the search path (note that the restricted shell will not execute such commands). You can give a full path name that begins with the root directory (as in `/bin/sort`), or a path name relative to the current directory, for example `bin/myfile`. In this last case, the shell looks in the current directory for a directory named `bin` and in that directory for `myfile`.

You can change the particular sequence of directories searched by resetting the **PATH** variable (page 923).

The shell remembers the location in the search path of each executed command (to avoid unnecessary **exec**s later). If the command was found in a ***relative directory*** (one whose name does not being with /), the shell must redetermine its location whenever the current directory changes. The shell forgets all remembered locations whenever you change the **PATH** variable or execute the **hash -r** command (page 932).

## Signals

The shell ignores **INTERRUPT** and **QUIT** signals for an invoked command if the command is terminated with a & (that is, if it is running in the background). Otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (see also the built-in **trap** command on page 934).

## The .profile File

When you log in, the shell is called to read your commands. Before it does that, however, it checks to see if a file named **/etc/profile** exists on the system, and if it does, it reads commands from it (this file should set variables needed by all users). After this, the shell looks for a file named **.profile** in your login directory. If it finds one, it executes commands from it. Finally, the shell is ready to read commands from your standard input.

## File-name Substitution

Command parameters are very often file names. You can automatically produce a list of file names as parameters on a command line by specifying a pattern that the shell matches against the file names in a directory.

Most characters in such a pattern match themselves, but you can also use some special pattern-matching characters in your pattern. These special characters are:

| | |
|---|---|
| * | Matches any string, including the null string. |
| ? | Matches any one character. |
| [ . . . ] | Matches any one of the characters enclosed in square brackets. |
| [! . . . ] | Matches any character *other than* one of the characters that follow the exclamation mark within square brackets. |

Inside square brackets, a pair of characters separated by a - (minus) specifies a set of all characters lexically within the inclusive range of that pair, according to the current collating sequence (see "**ctab**" on page 257). The **NLCTAB** environment variable controls the collating sequence.

The current collating sequence may group characters into ***equivalence classes*** for the purpose of defining the end points of a range of characters. For example, if the collating sequence defines the lexical order to be `AaBbCc . . .` and groups upper- and lower-case characters into equivalence classes, then all the following have the same effect: `[a-c]`, `[A-C]`, `[a-C]`, and `[A-c]`.

---

**Japanese Language Support Information**

A collating sequence in Japanese Language Support does not define equivalence classes for use in range expressions. To avoid unpredictable results when using a range expression to match a class of characters, use a ***character class expression*** rather than a standard range expression. For information about matching file names using character class expressions, see "File-name Substitution in Japanese Language Support" on page 917. See also the discussion of this topic included in the description of the command "File Name Substitution" on page 4.

---

Pattern matching has some restrictions. If the first character of a file name is a . (dot), it can be matched only by a pattern that literally begins with a dot. For example, * matches the file names `myfile` and `yourfile` but not the file names `.myfile` and `.yourfile`. To match these files, use a pattern such as the following:

```
.*file
```

If a pattern does not match any file names, then the pattern itself is returned as the result of the attempted match.

File and directory names should not contain the characters *, ?, [, or ] because they can cause infinite recursion (that is, infinite loops) during pattern-matching attempts.

---

**Japanese Language Support Information**

## File-name Substitution in Japanese Language Support

You can also use the following notation to match file names within a range indication:

[:*charclass*:]

This format instructs the system to match any single character belonging to *class*; the defined classes correspond to **ctype** subroutines. Following are the names of these classes:

| | |
|---|---|
| alnum | jalpha |
| alpha | jdigit |
| digit | jhira |
| lower | jkanji |
| print | jkata |
| punct | jparen |
| space | jpunct |
| upper | jspace |
| xdigit | jxdigit |

For example, the expression that matches any single kanji character would be the following:

```
[[:jkanji:]]
```

For additional information about character class expressions, see the discussion of this topic included in the description of the command "**ed**" on page 371.

---

## Shell Variables and Command-Line Substitutions

The shell has several mechanisms for creating variables (assigning a string value to a name). Certain variables, ***positional parameters*** and ***keyword parameters***, are normally set only on a command line. Other variables are simply names to which you or the shell can assign string values.

### *Positional Parameters*

When you run a shell procedure, the shell implicitly creates positional parameters that reference each word on the command line by its position on the command line. The word in position 0 (the procedure name), is called $0, the next word (the first parameter) is called $1, and so on up to $9. To refer to command line parameters numbered higher than 9, use the built-in **shift** command (page 934).

You can also assign values to these positional parameters explicitly by using the built-in **set** command (page 933).

**Notes:**

1. When an argument for a position is not specified, its positional parameter is set to null.
2. Positional parameters are global and can be passed to nested shell procedures.

### *User-defined Variables*

The shell also recognizes alphanumeric variables to which string values can be assigned. You assign a string value to a ***name***, as follows:

*name=string*

A name is a sequence of letters, digits, and underscores that begins with an underscore or a letter. To use the value that you have assigned to a variable, add a $. (dollar sign) to the beginning of its name. Thus $*name* yields the value *string*. Note that no blanks surround the = (equal sign) in an assignment statement. (Positional parameters cannot appear in a assignment statement; they can only be set as described earlier.) You can put more than one assignment on a command line, but remember: the shell performs the assignments from right to left.

If you surround *string* with quotation marks, either double or single (" " ' '), the shell does not treat blanks, tabs, semicolons, and new-line characters within it as word delimiters but imbeds them literally in the string.

If you surround *string* with double quotation marks (" "), the shell still recognizes variable names in the string and performs ***variable substitution***; that is, it replaces references to positional parameters and other variable names that are prefaced by $ with their corresponding values, if any. The shell also performs ***command substitution*** (see "Command Substitution" on page 925) within strings that are surrounded by double quotation marks.

If you surround *string* with single quotation marks (' '), the shell does no variable or command substitution within the string. The following sequence illustrates this difference:

```
    You: stars=*****
         asterisks1="Add $stars"
         asterisks2='Add $stars'
         echo $asterisks1
System: Add *****
    You: echo $asterisks2
System: Add $stars
```

The shell does not reinterpret blanks in assignments after variable substitution (see "Blank Interpretation" on page 930). Thus the following assignments result in `$first` and `$second` having the same value:

```
first='a string with embedded blanks'
second=$first
```

When you reference a variable, you can enclose the variable name (or the digit designating a positional parameter) in { } (braces) to delimit the variable name from any following string. In particular, if the character immediately following the name is a letter, digit, or underscore and the variable is not a positional parameter, then the braces are required:

```
    You: a='This is a'
         echo "${ a } n example"
Display: This is an example
    You: echo "$a test"
Display: This is a test
```

**Note:** The { } operator requires a space following the opening brace and a space preceding the closing brace.

See "Conditional Substitution" on page 920 for a different use of braces in variable substitutions.

### *A Command's Environment*

All the variables (with their associated values) that are known to a command at the beginning of its execution constitute its ***environment***. This environment includes variables that a command inherits from its parent process and variables specified as keyword parameters on the command line that calls the command.

The shell passes to its child processes the variables that have been named as arguments to the built-in **export** command. **export** places the named variables in the environments of both the shell and all its future child processes.

Keyword parameters are variable-value pairs that appear in the form of assignments, normally before the procedure name on a command line (but see also the **-k** flag on page 933). Such variables are placed in the environment of the procedure being called.

For example, given the following simple procedure that echoes the values of two variables (saved in a command file named key_command):

```
# key_command
echo $a $b
```

the following command lines produce the output shown:

```
   You: a=key1 b=key2 key_command
Display: key1 key2
   You: a=tom b=john key_command
Display: tom john
```

A procedure's keyword parameters are not included in the parameter count stored in $#.

A procedure can access the values of any variables in its environment; however, if it changes any of these values, these changes are not reflected in the shell environment. They are local to the procedure in question. To place these changes in the environment that the procedure passes to its child processes, you must export these values within that procedure.

To obtain a list of variables that have been made exportable from the current shell, enter:

```
export
```

(You will also get a list of variables that have been made **readonly**.) To get a list of name-value pairs in the current environment, enter:

```
env
```

### *Conditional Substitution*

Normally, the shell replaces $*variable* with the string value assigned to *variable*, if there is one. However, there is a special notation that allows ***conditional substitution***, depending on whether the variable is set and/or not null. By definition, a variable is ***set*** if it has ever been assigned a value. The value of a variable can be the null string, which you can assign to a variable in any one of the following ways:

```
A=
bcd=""
Efg=''
set '' ""
```

The first three of these examples assign the null string to each of the corresponding variable names. The last example sets the first and second positional parameters to the null string and unsets all other positional parameters.

The following is a list of the available expressions you can use to perform conditional substitution:

${ *variable-string* } If the variable is set, substitute the value of *variable* in place of this expression. Otherwise, replace this expression with the value of *string*.

${ *variable*:-*string* } If the variable is set and is not null, substitute the value of *variable* in place of this expression. Otherwise, replace this expression with the value of *string*.

${ *variable*=*string* }
If the variable is set, substitute the value of *variable* in place of this expression. Otherwise, set *variable* to *string* and then substitute the value of the *variable* in place of this expression. You cannot assign values to positional parameters in this fashion.

${ *variable*:=*string* }
If the variable is set and is not null, substitute the value of *variable* in place of this expression. Otherwise, set *variable* to *string* and then substitute the value of the *variable* in place of this expression. You cannot assign values to positional parameters in this fashion.

${ *variable*?*string* } If the variable is set, substitute the value of *variable* in place of this expression. Otherwise, display a message of the form:

*variable*: *string*

and exit from the current shell (unless the shell is the login shell). If you do not specify *string*, the shell displays the following message:

*variable*: parameter null or not set

${ *variable*:?*string* } If the variable is set and not null, substitute the value of *variable* in place of this expression. Otherwise, display a message of the form:

*variable*: *string*

and exit from the current shell (unless the shell is the login shell). If you do not specify *string*, the shell displays the following message:

*variable*: parameter null or not set

${ *variable*+*string* }
If the variable is set, substitute the value of *string* in place of this expression. Otherwise, substitute the null string.

${ *variable*:+*string* }
If the variable is set and not null, substitute the value of *string* in place of this expression. Otherwise, substitute the null string.

In conditional substitution, the shell does not evaluate *string* until it uses it as a substituted string, so that, in the following example, the shell executes the **pwd** command only if d is not set or is null:

```
echo ${ d:-`pwd` }
```

### *Variables Used by the Shell*

The shell uses the following variables. The shell sets some of them, and you can set or reset all of them:

**CDPATH** The search path for the **cd** (change directory) command (see the **PATH** variable in the following list for an explanation of search paths).

**HOME** The name of your ***login directory***, the directory that becomes the current directory upon completion of a login. The **login** program initializes this variable. The **cd** command uses the value of **$HOME** as its default value. If you use this variable in your shell procedures rather than using an explicit full path name, your procedures run even if your login directory is changed or if another user runs them.

**LIBPATH** The search path for shared libraries.

**LOGNAME** Your login name, marked **readonly** in the **/etc/profile** file.

**MAIL** The path name of the file used by the mail system to detect the arrival of new mail. If **MAIL** is set, the shell periodically checks the modification time of this file and displays the value of **$MAILMSG** if this time changes and the length of the file is greater than zero.

You should set **MAIL** in your **.profile** file. The value normally assigned to it by users of the **mail** command is **/usr/mail/$LOGNAME**.

**MAILCHECK** The number of seconds that the shell lets elapse before checking again for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If you set **MAILCHECK** to 0, the shell checks before each prompt.

**MAILPATH** A colon-separated list of file names (see **PATH**). If you set this parameter, the shell informs you of the arrival of mail in any of the files specified in the list. You can follow each file name by a % (percent sign) and a message to be displayed when mail arrives. Otherwise, the shell uses the value of **MAILMSG** or by default `"you have mail"`.

**Note:** When **MAILPATH** is set, these files are checked instead of the file set by **MAIL**. To check the files set by **MAILPATH** and the file set by **MAIL**, specify the **MAIL** file in your list of **MAILPATH** files.

**MAILMSG** The mail notification message. If you explicitly set **MAILMSG** to a null string (`MAIL=""`), no message is displayed.

**NLCTAB** Defines the collating sequence to use when sorting names and when character ranges occur in patterns. If absent, it may be taken form the parameter **NLFILE**. If both are absent, the American English collating sequence is used.

**Japanese Language Support Information**

When Japanese Language Support is installed, the default collating sequence, based on the character's value, is used. See "Overview of International Character Support" in *Managing the AIX Operating System* for further information.

**PATH** An ordered list of directory path names separated by colons. The shell searches these directories in the specified order when it looks for commands. A null string anywhere in the list represents the current directory.

**PATH** is normally initialized in the **/etc/profile** file, usually to **/bin:/usr/bin:/etc::**. You can reset this variable to suit your own needs. Thus if you wish to search your current directory first, rather than last, you would enter:

```
PATH=:/bin:/usr/bin:/etc
```

where, by definition, a null string is assumed in front of the leading colon. If you have a personal directory of commands (say, **$HOME/bin**) that you want searched before the standard system directories, set your **PATH** as follows:

```
PATH=$HOME/bin:/bin:/usr/bin:/etc::
```

The best place to set your **PATH** to something other than the default value is in your **.profile** file (see "The .profile File" on page 915). You cannot reset **PATH** if you are executing commands under the restricted shell.

**PS1** The string to be used as the primary system prompt. An interactive shell displays this prompt string when it expects input. The default value of PS1 is "$" (a $ followed by a blank).

**PS2** The value of the secondary prompt string. If the shell expects more input when it encounters a new-line character in its input, it prompts with the value of PS2. The default value of PS2 is "> " (a > followed by a blank).

**IFS** The characters that are ***internal field separators*** (the characters that the shell uses during blank interpretation, see "Blank Interpretation" on page 930). The shell initially sets **IFS** to include the blank, tab, and new-line characters.

**SHACCT** The name of a file that you own. If this parameter is set, the shell writes an accounting record in the file for each shell script executed. You can use accounting programs such as **acctcom** and **acctcms** to analyze the data collected.

**SHELL** A path name whose simple part (the part after the last /) contains the letter r if you want the shell to become restricted when invoked. This should be set and exported by the **$HOME/.profile** file of each restricted login.

**TIMEOUT** A number of minutes. After the shell displays its prompt, you have **TIMEOUT** minutes to enter a command. If you fail to do so, the shell exits; in the login shell, such an exit is a logoff. Setting **TIMEOUT** to 0 inhibits automatic logoff.

### *Predefined Special Variables*

Several variables have special meanings; the following are set *only* by the shell:

$# The number of positional parameters passed to the shell, not counting the name of the shell procedure itself. $# thus yields the number of the highest-numbered positional parameter that is set. One of the primary uses of this variable is to check for the presence of the required number of arguments.

$? The exit value of the last command executed. Its value is a decimal string. Most UNIX commands return 0 to indicate successful completion. The shell itself returns the current value of $? as its exit value.

$$ The process number of the current process. Because process numbers are unique among all existing processes, this string of up to five digits is often used to generate unique names for temporary files. The following example illustrates the recommended practice of creating temporary files in a directory used only for that purpose:

```
temp=$HOME/temp/$$
ls >$temp
    .
    .
    .
rm $temp
```

$! The process number of the last process run in the background (using the & terminator). Again, this is a string of up to five digits.

$- A string consisting of the names of the execution flags (page 933) currently set in the shell.

### *Command Substitution*

To capture the output of any command as an argument to another command, place that command line within grave accents (` `). This concept is known as command substitution. The shell first executes the command or commands enclosed within the grave accents, and then replaces the whole expression, grave accents and all, with their output. This feature is often used in assignment statements:

```
today=`date`
```

This statement assigns the string representing the current date to the variable `today`. The following assignment saves, in the variable `files`, the number of files in the current directory:

```
files=`ls | wc -l`
```

You can enclose any command that writes to standard output in grave accents. You can nest command substitutions as long as you quote the inside sets of grave accents with a preceding \ (backslash):

```
logmsg=`echo Your login directory is \`pwd\"
```

You can also give values to shell variables indirectly by using the built-in **read** command. **read** takes a line from standard input (usually your keyboard), and assigns consecutive words on that line to any variables named. For example,

```
read first init last
```

will take an input line of the form:

```
J. Q. Public
```

and have the same effect as if you had typed:

```
first=J.    init=Q.    last=Public
```

**read** assigns any excess words to the last variable.

## Quoting Mechanisms

Many characters have a special meaning to the shell; sometimes you want to conceal that meaning. Single (' ') and double (" ") quotation marks surrounding a string or a backslash (\) before a single character provide this function in somewhat different ways.

Within single quotation marks, all characters (except the single quotation character itself), are taken literally, with any special meaning removed. Thus:

```
stuff='echo $? $*; ls * | wc'
```

results only in the literal string `echo $? $*; ls * | wc` being assigned to the variable `stuff`; the `echo`, `ls`, and `wc` commands are not executed, nor are the variables $? and $* and the special character * expanded by the shell.

Within double quotation marks, the special meaning of certain characters (the $, `, and ") does persist, while all other characters are taken literally. Thus, within double quotation marks, command and variable substitution takes place. In addition, the quotation marks do not affect the commands within a command substitution that is part of the quoted string, so characters there retain their special meanings.

Consider the following sequence:

```
   You: ls *
Display: file1
         file2
         file3
   You: message="This directory contains `ls * ` "
        echo $message
Display: This directory contains file1 file2 file3
```

This shows that the * special character inside the command substitution was expanded.

To hide the special meaning of $, `, and " within double quotation marks, precede these characters with a \ (backslash). Outside of double quotation marks, preceding a character with \ is equivalent to placing it within single quotation marks. Hence, a \ immediately preceding the new-line character (that is, a \ at the end of the line) hides the new-line character and allows you to continue the command line on the next physical line.

## Redirection of Input and Output

In general, most commands do not know or care whether their input or output is associated with the keyboard, the display screen, or a file. Thus a command can be used conveniently either at the keyboard or in a pipeline.

### *Standard Input and Standard Output*

When a command begins running, it usually expects that three files are already open: ***standard input***, ***standard output***, and ***diagnostic output*** (sometimes called ***error output*** or ***standard error output***). A number called a ***file descriptor*** is associated with each of these files as follows:

| | |
|---|---|
| File descriptor 0 | Standard input |
| File descriptor 1 | Standard output |
| File descriptor 2 | Diagnostic (error) output |

A child process normally inherits these files from its parent; all three files are initially assigned to the work station (0 to the keyboard, 1 and 2 to the display). The shell permits them to be redirected elsewhere before control is passed to a command. Any argument to the shell in the form *<file* or *>file* opens the specified file as the standard input or output, respectively. In the case of output, this process destroys the previous contents of *file*, if it already exists. An argument in the form *>>file* directs the standard output to the end of *file*, thus allowing you to add data to it without destroying its existing contents. If *file* does not exist, the shell creates it.

Such redirection arguments are subject only to variable and command substitution; neither blank interpretation nor pattern matching of file names occurs after these substitutions. Thus:

```
echo 'this is a test' > *.ggg
```

produces a one-line file named `*.ggg` (a disastrous name for a file), and:

```
cat < ?
```

produces an error message, unless you have a file named ? (also a bad choice for a file name).

### *Diagnostic and Other Output*

Diagnostic output from UNIX commands is normally directed to the file associated with file descriptor 2. You can redirect this error output to a file by immediately preceding either output redirection arrow (> >>) with a 2 (the number of the file descriptor). For example, the following line adds error messages from the **cc** command to the file ERRORS:

```
cc testfile.c 2>> ERRORS
```

Note that there must be no blanks between the file descriptor and the redirection symbol; otherwise, the shell interprets the number as a separate argument to the command.

You can also use this method to redirect the output associated with any of the first 10 file descriptors (numbered 0 through 9) so that, for instance, if a command (`cmd`) writes to file descriptor 9 (although this is not a recommended programming habit), you can capture that output in a file `savedata` as follows:

```
cmd 9> savedata
```

If a command writes to more than one output, you can independently redirect each one. Suppose that a command directs its standard output to file descriptor 1, directs its error output to file descriptor 2, and builds a data file on file descriptor 9. The following command line redirects each of these outputs to a different file:

```
cmd > standard 2> error 9> data
```

### *Inline Input Documents*

Upon seeing a command line of the form:

`cmd << eofstring`

where *eofstring* is any string that does not contain any pattern-matching characters, the shell takes the subsequent lines as the standard input of `cmd` until it reads a line consisting of only *eofstring* (possibly preceded by one or more tab characters). The lines between the first *eofstring* and the second are frequently referred to as a ***here document***. If a - (minus) immediately follows the <<, the shell strips leading tab characters from each line of the input document before it passes the line to the command.

The shell creates a temporary file containing the input document and performs variable and command substitution on its contents before passing it to the command. It performs pattern matching on file names that are a part of command lines in command substitutions. If you want to prohibit all substitutions, quote any character of *eofstring*:

`cmd << \eofstring`

The here document is especially useful for a small amount of input data that is more conveniently placed in the shell procedure rather than kept in a separate file (such as editor "scripts"). For instance, you could enter:

```
cat <<- xyz
                                This message will be shown on the
                                display with leading tabs removed.
xyz
```

This feature is most useful in shell procedures. Note that inline input documents *cannot* appear within grave accents (command substitution).

### *Input and Output Redirection Using File Descriptors*

As discussed previously, a command occasionally directs output to some file associated with a file descriptor other than 1 or 2. The shell also provides a mechanism for creating an output file associated with a particular file descriptor. By entering:

*fd1*>&*fd2*

where *fd1* and *fd2* are valid file descriptors, you can direct the output that would normally be associated with file descriptor *fd1* to the file associated with *fd2*. The default value for *fd1* and *fd2* is 1 (standard output). If, at execution time, no file is associated with *fd2*, then the redirection is void. The most common use of this mechanism is to ***direct*** standard error output to the same file as standard output, as follows:

```
cmd 2>&1
```

If you want to ***redirect*** both standard output and standard error output to the same file, enter:

```
cmd > file  2>&1
```

The order here is significant. First, the shell associates file descriptor 1 with `file`; then it associates file descriptor 2 with the file that is currently associated with file descriptor 1. If you reverse the order of the redirections, standard error output will go to the display and standard output would go to `file` because at the time of the error output redirection, file descriptor 1 was still associated with the display.

You can also use this mechanism to redirect standard input. You could enter:

*fda*<&*fdb*

to cause both file descriptors to be associated with the same input file. For commands that run sequentially, the default value of *fda* and *fdb* is 0 (standard input). For commands that run asynchronously (commands terminated by &), the default value of *fda* and *fdb* is **/dev/null**. Such input redirection is useful for commands that use two or more input sources.

## *Summary of Redirection Options*

The following can appear anywhere in a simple command or can precede or follow a command, but they are not passed to the command:

| | |
|---|---|
| <*file* | Use *file* as standard input. |
| >*file* | Use *file* as standard output. Create the file if it does not exist; otherwise truncate it to zero length. |
| > >*file* | Use *file* as standard output. Create the file if it does not exist; otherwise add the output to the end of the file. |
| < <[-]*eofstr* | Read as standard input all lines from *eofstr* up to a line containing only *eofstr* or up to an end-of-file character. If any character in *eofstr* is quoted, the shell does not expand or interpret any characters in the input lines; otherwise, it performs variable and command substitution and ignores a quoted new-line character (\new-line). Use a \ to quote characters within *eofstr* or within the input lines.<br><br>If you add a - (minus) to < < then all leading tabs are stripped from *eofstr* and from the input lines. |
| <&*digit* | Associate standard input with file descriptor *digit*. |
| >&*digit* | Associate standard output with file descriptor *digit*. |
| <&- | Close standard input. |

>&- Close standard output.

The restricted shell does not allow the redirection of output.

## Blank Interpretation

After the shell performs variable and command substitution, it scans the results for internal field separators (those defined in the shell variable **IFS**, see page 924). It splits the line into distinct words at each place it finds one of these characters. It retains explicit null arguments ("" '') and discards implicit null arguments (those resulting from parameters that have no values).

## Control Commands

The shell provides several flow control commands that are useful in creating shell procedures:

**for** *name* [ **in** *word* . . . ]

**do** *list*
**done** For each *word*, sets *name* to *word* and executes the commands in *list*. If you omit **in** *word* . . ., then the **for** command executes *list* for each positional parameter that is set. Execution ends when there are no more words in the word list.

**case** *word* **in**
*pattern* [|*pattern*] . . . **)** *list*;;

[ .
.
.
*pattern* [|*pattern*] . . . **)** *list*;;]

**esac** Executes the commands in the *list* associated with the first *pattern* that matches *word*. Uses the same character-matching notation in patterns that you use for file-name substitution (see "File-name Substitution" on page 916), except that you do not need to match explicitly a slash, a leading dot, or a dot immediately following a slash.

**if** *list*
**then** *list*
[**elif** *list* ] . . .
[**else** *list*]
**fi** Executes the *list* following the **if** keyword. If it returns a zero exit value, execute the *list* following the first **then**. Otherwise, execute the *list* following **elif** (if there is an **elif**), and if its exit value is zero, execute the next **then**. Failing that, execute the *list* following the **else**.

If no **else** *list* or **then** *list* is executed, the **if** command returns a zero exit value.

**while** *list*
**do** *list*
**done**
: Executes the *list* following the **while**. If the exit value of the last command in the list is zero, executes the *list* following **do**. Continue looping through the *list*s until the exit value of the last command in the **while** *list* is nonzero. If no commands in the **do** *list* are executed, the **while** command returns a zero exit value.

**until** *list*
**do** *list*
**done**
: Executes the *list* following the **until**. If the exit value of the last command in the list is nonzero, executes the *list* following **do**. Continues looping through the *list*s until the exit value of the last command in the **until** *list* is zero. If no commands in the **do** *list* are executed, the **until** command returns a zero exit value.

**(***list***)**
: Executes the commands in *list* in a subshell.

**{** *list*; **}**
: Executes the commands in *list* in the current shell process (does not spawn a subshell).

*name* **() {** *list*; **}**
: Defines a function that is referenced by *name*. The body of the function is the *list* of commands between the braces.

The following words are recognized only as the first word of a command and when not quoted:

**if then else elif fi case esac for while until do done { }**

## Built-in Commands

**:**
: Does nothing. This null command returns a zero exit value.

**.** *file*
: Reads and executes commands from *file* and returns. Does not spawn a subshell. The search path specified by **PATH** is used to find the directory containing *file*.

**break** [*n*]
: Exits from the enclosing **for**, **while**, or **until** loop, if any. If *n* is specified, then breaks *n* levels.

**continue** [*n*]
: Resumes the next iteration of the enclosing **for**, **while**, or **until** loop. If *n* is specified, resumes at the *n*th enclosing loop.

**cd** [*dir*]
: Changes the current directory to *dir*. The value of the shell variable **HOME** is the default *dir*. The shell variable **CDPATH** defines the search path for the directory containing *dir*. Alternative directory names appear in a colon-separated list. A null path name specifies the current directory (which is the default path). This null path name can

appear immediately after the equal sign in the assignment or between the colon delimiters anywhere else in the path list. If *dir* begins with a slash, the shell does not use the search path. Otherwise, the shell searches each directory in the path. **cd** cannot be executed by the restricted shell.

**dirstyle** Indicates whether directories should be interpreted in either raw or System V format. A + flag indicates path names from remote file systems should be converted to System V format. A - flag indicates path names from remote file systems should not be converted to System V.

> **Note:** Use this only when reading the contents of a directory that is not formatted in System V. Otherwise, use **opendir**, **readdir**, and **closedir** library functions.

**echo** [*arg . . .* ] Writes arguments to standard output. See "**echo**" on page 369 for a discussion of its usage and parameters.

**eval** [*arg . . .* ] Reads arguments as input to the shell and executes the resulting command(s).

**exec** [*arg . . .* ] Executes the command specified by argument in place of this shell without creating a new process. Input and output arguments can appear and, if no other arguments appear, cause the shell input or output to be modified (not a good idea with your login shell).

**exit** [*n*] Causes a shell to exit with the exit value specified by *n*. If you omit *n*, the exit value is that of the last command executed (**Ctrl-D** will also cause a shell to exit). The value of *n* can be from 0 to 255, inclusive.

**export** [*name . . .* ]
Marks the specified *name*s for automatic export to the environments of subsequently executed commands. If you do not specify *name*s, **export** displays a list of all names that are exported in this shell. You *cannot* export function names.

**hash** [**-r**] [*name . . .* ]
For each *name*, finds and remembers the location in the search path of the command specified by *name*. The **-r** flag causes the shell to forget all locations. If you do not specify the flag or any *name*s, the shell displays information about the remembered commands. In this information, ***hits*** is the number of times a command has been run by the shell process. ***Cost*** is a measure of the work required to locate a command in the search path. There are certain situations that require that the stored location of a command be recalculated (for example, the location of a relative path name when the current directory changes). Commands for which that might be done are indicated by an asterisk next to the hits information. Cost is incremented when the recalculation is done.

**newgrp** [*arg* . . . ]
Executes the **newgrp** command in the current shell process. See "**newgrp**" on page 689 for a discussion of command options.

**pwd** Displays the current directory. See "**pwd**" on page 800 for a discussion of command options.

**read** [*name* . . . ] Reads one line from standard input. Assigns the first word in the line to the first *name*, the second word to the second *name*, and so on, with leftover words assigned to the last *name*. This command returns a 0 unless it encounters an end of file.

**readonly** [*name* . . . ]
Marks the specified *name*s **readonly**. The values of these *name*s cannot be reset. If you do not specify any *name*s, **readonly** displays a list of all **readonly** names.

**return** [*n*] Cause a function to exit with a return value of *n*. If you do not specify *n*, the function returns the status of the last command executed in that function. This command is valid only when executed within a shell function.

**set** [*flag* . . . [*arg*] . . . ]

- **-a** Marks for export all variables that are modified or changed.
- **-e** Exits immediately if a command exits with a nonzero exit value.
- **-f** Disables file-name substitution.
- **-h** Locates and remembers the commands called within functions as the functions are defined (normally these commands are located when the function is executed—see the **hash** command on page 932).
- **-k** Places all keyword parameters in the environment for a command, not just those that precede the command name.
- **-n** Reads commands but do not execute them.
- **-t** Exits after reading and executing one command.
- **-u** Treats an unset variable as an error when performing variable substitution.
- **-v** Displays shell input lines as they are read.
- **-x** Displays commands and their arguments as they are executed.
- **--** Does not change any of the flags. This is useful in setting $1 to a string beginning with a - (minus).

Using a + (plus) rather than a - (minus) unsets flags. You can also specify these flags on the shell command line. The special variable $- contains the current set of flags.

Any arguments to **set** are positional parameters and are assigned, in order, to $1, $2, and so on. If you do not specify flags or parameters, **set** displays all names.

**shift** [*n*]
Shifts command line arguments to the left; that is, reassign the value of the positional parameters by discarding the current of value of $1 and assigning the value of $2 to $1, of $3 to $2, and so on. If there are more than 9 command line arguments, the tenth is assigned to $9 and any that remain are still unassigned (until after another **shift**). If there are 9 or fewer arguments, a **shift** unsets the highest-numbered positional parameter.

$0 is never shifted. The command **shift** *n* is a shorthand notation for *n* consecutive shifts. The default value of *n* is 1.

**test** *expr* | [ *expr* ]
Evaluates conditional expressions. See "**test**" on page 1064 for a discussion of command options.

**times**
Displays the accumulated user and system times for processes run from the shell.

**trap** [*arg*] [*n*] . . .

Runs the command specified by *arg* when the shell receives signal(s) *n*. (Note that the shell scans *parm* once when the trap is set and once when the trap is taken). **trap** commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective.

If you do not specify an *arg*, then all trap(s) *n* are reset to their current values. If *arg* is the null string, then this signal is ignored by the shell and by the commands it invokes. If *n* is 0, then *arg* is executed on exit from the shell. If you specify no *arg* and no *n*, **trap** displays a list of commands associated with each signal number.

**type** [*name* . . . ]
For each *name*, indicates how the shell would interpret it as a command name.

**ulimit** [**-b** [*m*]] [**-f**] [*n*] [**-s** [*m*]]
Sets or queries size limits. The **-b** flag sets the break value to *m*. This limits the size of data segment to *m* pages. If you specify **-b** with no *m*, **ulimit** displays the current break value. The **-f** flag imposes a size limit of *n* blocks on files written by the child processes (files of any size can be read). Without *n*, **ulimit** displays the current limit (this is the default action of **ulimit**).

**Notes:**

1. Since the shell rounds *n* down to the nearest cluster size, it is best to make it a multiple of 4 (for example, specifying values of 1, 2, or 3 for *n* all result in a size limit of 0).
2. Any user can decreased this limit, but only a user operating with superuser authority can increase the limit.

The **-s** flag imposes a size limit of *m* pages on the stack. If you specify **-s** with no *m*, **ulimit** displays the current stack size limit.

**umask** [*nnn*] Sets the user file-creation mask to *nnn* (see the **umask** system call). If you omit *nnn*, **umask** displays the current value of the mask.

**unset** [*name . . .* ]
For each *name*, removes the corresponding variable or function. The variables **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS** cannot be unset.

**wait** [*n*] Waits for the child process whose process number is *n* to end and reports its termination status. If you do not specify *n*, then the shell waits for all currently active child processes and the return value is 0.

## Running the Shell

The **sh** command can be run either as a ***login shell*** or as a login shell subshell under the login shell. Only the **login** command can call **sh** as a login shell. It does this by using a special form of the **sh** command name: `-sh`. When called with an initial - (minus), the shell first reads and runs commands found in the system **profile** file and your **$HOME/.profile**, if one exists. It then accepts commands as described in the following discussion of flags. Once logged in and working under a login shell, you can call **sh** with the command name `sh`. This command runs a subshell, a second shell running as a child of the login shell.

## Restricted Shell

The restricted shell, **Rsh**, is used to set up login names and environments whose capabilities are more controlled than those of the standard shell. The actions of **Rsh** are identical to those of **sh**, except that the following are not allowed:

- Changing directory (see "**cd**" on page 150)
- Setting the value of **$PATH**
- Specifying path or command names containing /
- Redirecting output (> and > >).

The restrictions above are enforced after **.profile** is interpreted, meaning that the restrictions can override settings originally set in **.profile**.

When a command to be run is found to be a shell procedure, **Rsh** starts **sh** to run it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and run permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing setup actions and leaving the user in an appropriate directory (probably not the login directory).

When called with the name **-Rsh**, **Rsh** reads the user's **.profile** (from $HOME/**.profile**). It acts as the standard **sh** while doing this, except that an interrupt causes an immediate exit instead of a return to command level.

## Flags

The following flags are interpreted by the shell only when you call it. Note that unless you specify either the **-c** or **-s** flag, the shell assumes that the next parameter is a command file (shell procedure). It passes anything else on the command line to that command file (see "Positional Parameters" on page 918).

**-c** *cmdstring*
: Runs commands read from *cmdstring*. The shell does not read additional commands from standard input when you specify this flag.

**-i**
: Makes the shell interactive, even if input and output are not from a work station. In this case the shell ignores the **TERMINATE** signal (so that **kill 0** does not stop an interactive shell) and traps an **INTERRUPT** (so that you can interrupt **wait**). In all cases, the shell ignores the **QUIT** signal. (See the **signal** system call in *AIX Operating System Technical Reference* and "**kill**" on page 552 for more information about signals.)

**-r**
: Creates a restricted shell (the same as running **Rsh**).

**-s**
: Reads commands from standard input. Any remaining parameters specified are passed as positional parameters to the new shell. Shell output is written to standard error, except for the output of built-in commands (see "Built-in Commands" on page 931).

The remaining flags and parameters are described in the built-in **set** command on page 933.

## Files

/etc/environment
/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

## Related Information

The following commands: "**cd**" on page 150, "**echo**" on page 369, "**env**" on page 393, "**login**" on page 584, "**newgrp**" on page 689, "**pwd**" on page 800, "**test**" on page 1064 and "**umask**" on page 1110.

The **dup**, **exec**, **fork**, **fullstat**, **pipe**, **signal**, **ulimit**, and **umask** system calls and the **a.out**, **.profile**, and **environ** files in *AIX Operating System Technical Reference.*

"Using the Shell with Processes" and "Advanced Shell Features" in *Using the AIX Operating System.*

"Overview of International Character Support" in *Managing the AIX Operating System.*

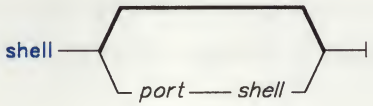
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

# shell

## Purpose

Executes a shell in a user's login environment

## Syntax




AJ2FL133

## Description

The **shell** command returns a user to an environment that is equivalent to the user's login environment. This command issues a **frevoke** system call to end all processes accessing the port. It re-establishes the terminal modes and environment variables then runs the user's login shell or the specified shell.

Only a person with superuser authority can specify *port* and *shell*. This command ignores any other arguments. When *port* and *shell* are not specified, the shell is found in **/etc/passwd** and the terminal is the one from which this command is issued.

**Japanese Language Support Information**

If Japanese Language Support is installed on your system, this command is not available.

## File

| | |
|---|---|
| /etc/passwd | Specifies the user's login shell. |

## Related Information

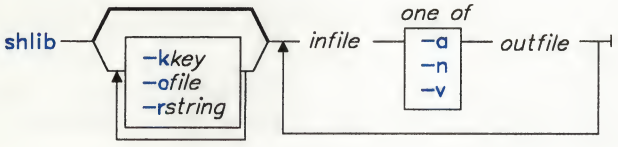
See the section on trusted path management in *Managing the AIX Operating System*.

# shlib

## Purpose

Creates a shared library.

## Syntax




OL805448

## Description

The **shlib** command creates a shared library from a set of unshared object and/or archive files. The shared library it creates has two parts:

- A single shared library text image that contains the code, and only the code, from all of the input files.
- Modified archive or object files that refer to the text image, each of which corresponds to one of the **shlib** input files.

By default, **shlib** uses the name of the first input file to generate the shared library key. It does this by removing any directory path from the file name and, if the file name does not contain a suffix, adding the suffix **.yyddd**, where **yy** is the last two digits of the current year and **ddd** is the number of the day. **shlib** puts this key in each of the modified output files. By default, it also uses this key as the name of the shared library text image, which contains the shared library key in a form that can be found by the **what** command.

The **shlib** command transforms each input file specified on the command line and copies it to (or verifies it against) an output file. Each output object module differs from the corresponding input object module in that the text portion has been removed and added to the end of the shared library text image. **shlib** also appends the shared library key to each output module and marks its **a.out** header so the **ld** command recognizes that the file refers to a shared library and relocates references appropriately.

Once you create an archive for a shared library, you can use the **ar** command to replace individual object files. The new object files will not refer to the shared library, thus allowing you to patch shared libraries.

The **shlib** command can process all **cc** and **f77** programs. Other programs must have **KCALL** relocation entries as the only external references within the text portion (see the **a.out** file in *AIX Operating System Technical Reference*). **KCALL** relocation entries are replaced by **balax** instructions to location 0xC00, which contain a code fragment to continue calls across segments. That code requires that register 0 point to the constant pool of the called routine, the first entry of which is the entry point of the invoked routine.

## Flags

| | |
|---|---|
| **-a** *outfile* | Adds new object modules in archives to *outfile*. This lets you add functions to a shared library, replacing the shared library text image without relinking programs that refer to it. The entire shared library image must be rebuilt. |
| **-k***key* | Uses the shared library *key* in the output object modules to refer to the shared library text image. If *key* does not contain a . (period), a suffix in the form **.yyddd** is added. |
| **-n** *outfile* | Makes a new output file for each input file. |
| **-o***file* | Assigns the name *file* to the shared library text image. **shlib** always makes a new shared library text image, replacing the old one. |
| **-r***string* | Adds *string* to the end of the shared library key in the **what** string. This only applies to the text image file. |
| **-v** *outfile* | Verifies components of a changed shared library, thus ensuring that the resulting object files are the same as the files previously created by **shlib**. Changes can be made to C source code as long as references to external names are not added, deleted, or rearranged and no floating point, string, or static initial values are modified. |

## Examples

1. To create a new shared text image:

```
shlib -kclibs -r"C shared library text" \
   /lib/libc.a -n /lib/libcs.a /lib/librts.a -n /lib/librtss.a
```

   This creates shared libraries `libcs.a` and `librtss.a` and the text image file `clibs.86133` (86133 indicates that this file was produced on 13 May 1986).

2. To modify an existing shared text image:

```
shlib -kclibs -r"C shared library text" \
   /lib/libc.a -a /lib/libcs.a  /lib/libm.a -n /lib/libms.a
```

   This updates the shared library `libcs.a`, creates the shared library `libms.a`, and updates the shared text image `clibs.86133`. (`libc.a` may contain new members that are added to the shared library and the text image.)

3. To create a shared text image with a different name than its key:

```
shlib -kclibs -octext_image -r"C shared library text" \
   /lib/libc.a -n /lib/libcs.a  /lib/librts.a -n /lib/librtss.a
```

   This produces a text image file named `ctext_image`. Note that you must use the **-k** flag when you compile programs that use this text image, since the key and the file name are not the same:

```
cc  myproj.c -kclibs:ctext_image -lrtss -lcs
```

## Related Information

The following command: "**ld**" on page 557.

The **profil** system call, **monitor** subroutine and **a.out** file in *AIX Operating System Technical Reference*.

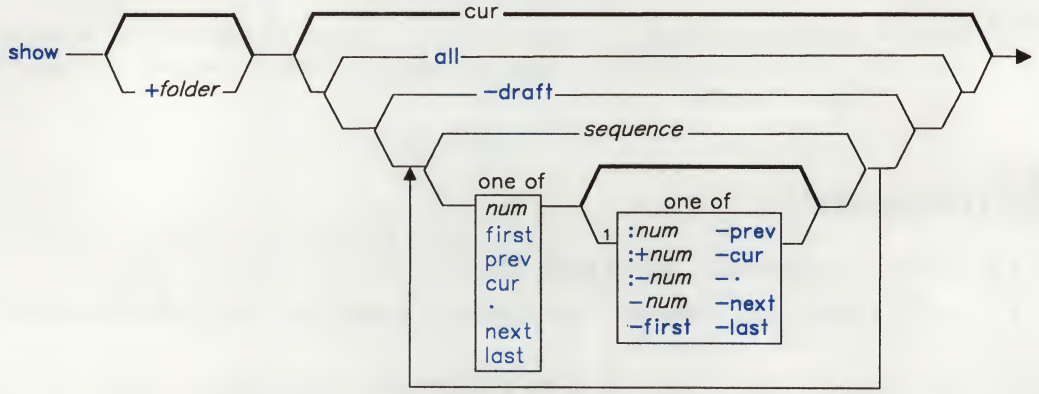
The discussion of shared libraries in *AIX Operating System Programming Tools and Interfaces*.

# show

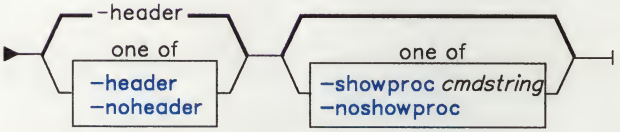
## Purpose

Shows messages.

## Syntax




AJ2FL207





AJ2FL208

[1] Do not put a blank between these items.

OL805308

## Description

The **show** command is used to display a list of messages. **show** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **show** command sends a listing of each of the specified messages to standard output. If standard output is not a display, **show** lists each message with a one-line header and two separation lines.

The **show** command invokes a program to perform the listing. The system default is **/bin/pg**. You can define your own default with the **showproc:** entry in **$HOME/.mh_profile**. If you set **showproc:** entry to **mhl**, **show** calls an internal **mhl** routine instead of the **mhl** command. You can also specify the program to perform a listing in the *cmdstring* of the **-showproc** flag.

The **show** command passes any flags that it does not recognize to the program performing the listing. Thus, you can specify flags for the listing program, as well as the flags described in this command section.

If the **Unseen-Sequence** entry is present in **$HOME/.mh_profile** and is not empty, **show** removes each of the messages shown from each sequence named by the profile entry.

## Flags

**-draft** Shows the file *user_mh_directory*/**draft** if it exists.

**+***folder msgs* Specifies the messages that you want to show. *msgs* can be several messages, a range of messages, or a single message. You can use the following message references when specifying *msgs*:

| | | |
|---|---|---|
| *num* | **first** | **prev** |
| **cur** | **.** | **next** |
| **last** | **all** | *sequence* |

The default message is the current message in the current folder. If several messages are specified, the last message shown becomes the current message.

**-header** Displays a one-line description of the message being shown. The description includes the folder name and the message number. If you show more than one message, this flag does not produce message headers. This flag is the default.

**-help** Displays help information for the command.

**-noheader** Does not display a one-line description of each message being shown.

**-noshowproc** Uses **/bin/cat** to perform the listing.

**-showproc** *cmdstring* Uses the specified command string to perform the listing.

## Profile Entries

| | |
|---|---|
| **Current-Folder:** | Sets your default current folder. |
| **Path:** | Specifies your *user_mh_directory*. |
| **showproc:** | Specifies the program used to show messages. |
| **Unseen-Sequence:** | Specifies the sequences used to keep track of your unseen messages. |

## Files

| | |
|---|---|
| $HOME/.mh_profile | The MH user profile. |
| *user_mh_directory*/draft | The draft file. |

## Related Information

The following commands: "**mhl**" on page 643, "**next**" on page 694, "**pick**" on page 748, "**prev**" on page 765, "**scan**" on page 871, "**sendmail**" on page 897.

The **mh-format**, **mh-mail**, and **mh-profile** files in *AIX Operating System Technical Reference*.

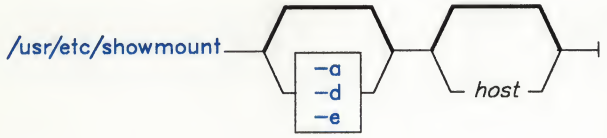
"Overview of the Message Handling Package" in *Managing the AIX Operating System*.

# showmount

## Purpose

Shows all file systems that are mounted remotely in an NFS environment.

## Syntax




OL805502

## Description

The **showmount** command lists the clients that have mounted a file system from an NFS server. This information is maintained by the network daemon **mountd** which runs continuously on the server. The daemon saves this information to the file **/etc/rmtab** for use if the server crashes.

The default value for *host* is the value returned by the command **hostname**.

**Japanese Language Support Information**

If Japanese Language Support is installed on your system, this command is not available.

## Flags

**-a** Displays all remote mounts in the format *hostname:directory*, where *hostname* specifies the name of the client and *directory* specifies the root of the mounted file system.

**-d** Lists directories that have been remotely mounted by clients.

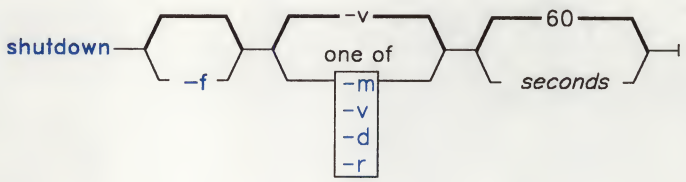
**-e** Prints the list of exported file systems.

# shutdown

## Purpose

Ends system operation.

## Syntax




OL805215

## Description

The **shutdown** command brings the AIX Operating System from distributed mode to multiuser mode, from multiuser mode to maintenance mode, or halts the operating system completely. You can run **shutdown** only if you are a member of the system group or if you have superuser authority.

During the default shutdown, users are notified (by a **wall** command) of the impending system shutdown with the message: `THE SYSTEM IS COMING DOWN NOW`. However, the shutdown is not complete until the user receives the message: `...shutdown completed...`. Do not attempt to reboot the system (**Ctrl-Alt-Pause**) or turn off the system before this message is displayed; otherwise, file system damage may result.

During the shutdown, the **hold** command prevents any new logins. After the specified number of *seconds* (60 by default), the system stops the accounting and error-logging processes. **shutdown** then runs the **killall** command to end any remaining processes and runs the **sync** command to flush all memory resident disk blocks. Finally, it unmounts the file systems and sends the appropriate signal to **init**. These signals are:

| | |
|---|---|
| **SIGINT** | Maintenance mode |
| **SIGTERM** | Virtual machine halt. |

**Note:** Users who have files open on the node that is running **shutdown**, but who are not logged in to that node are not notified about the shutdown.

If there are no other users on the system and you want to shutdown the system quickly, you may want to use **shutdown -f**. This option bypasses messages to users and brings the system down as quickly as possible.

If you request a complete halt to the operating system, **shutdown** kills all processes, unmounts all file systems, and sends **init** the **SIGTERM** signal.

**Warning:** If you are bringing the system down to maintenance mode, you must run **shutdown** from the root directory to ensure that it can cleanly unmount the file systems.

## Flags

**-d** Brings the system down from a distributed mode to a multiuser mode.

**-f** Does a fast shutdown, bypassing the messages to other users and bringing the system down as quickly as possible. If you do not specify this flag, **shutdown** sends a message to each logged-in user and waits a certain amount of time before bringing the system down, to allow each user to log off cleanly.

**-m** Brings the system down to maintenance mode. From maintenance mode, you can return to multiuser mode.

**-r** Causes the system to automatically reboot (**Ctrl-Alt-Pause**) after the user receives the message `..shutdown completed`.

**-v** Halts the operating system completely.

## Examples

1. To tell the operating system you are about to turn off the machine:

   ```
   shutdown
   ```

   This shuts down the system, waiting 60 seconds before stopping the user processes and the **init** process.

2. To give users and the system more time to finish what they are doing:

   ```
   shutdown  -m  120  *
   ```

   This brings the system down from multiuser mode to maintenance mode after waiting 120 seconds.

## Files

/etc/shutdown.sh
/etc/fshutdown.sh

## Related Information

The following commands: "**acct/***" on page 13, "**errstop**" on page 404, "**init**" on page 521, "**kill**" on page 552, and "**killall**" on page 555,

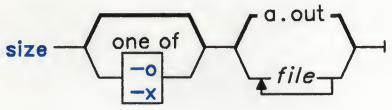
The **dsstate** and **signal** system calls in *AIX Operating System Technical Reference.*

# size

## Purpose

Displays the section sizes of common object files.

## Syntax




OL805216

## Description

The **size** command writes to the standard output the number of bytes required by the text, initialized data, and uninitialized data, along with their sum for each *file*. The default *file* is **a.out**.

## Flags

The output is in decimal notation unless you change the output with the following flags:

**-o** Writes in octal notation.

**-x** Writes in hexadecimal notation.

## Examples

1. To display the size of **a.out** in decimal:

   ```
   size
   ```

   This displays the size in bytes of the executable file **a.out**. The size of each section of the object file is given, followed by the total. The three sections are program text, data, and bss (uninitialized data). The values are in decimal.

2. To display the size of an object file in octal:

   ```
   size  -o  driver.o
   ```

   This displays the size of the object file `driver.o` in octal.

3. To display the size of several object files in hexidecimal:

   ```
   size  -x  *.o
   ```

   This displays in hexadecimal the size of each file in the current directory ending with `.o`.

## Related Information

The following commands: "**ar**" on page 55, "**as**" on page 61, "**cc**" on page 140, "**dump**" on page 366, "**ld**" on page 557, "**nm**" on page 705, and "**strip**" on page 1017.

The discussion of the **a.out** and **ar** files in *AIX Operating System Technical Reference.*

# skulker

## Purpose

Cleans up file systems by removing unwanted files.

## Syntax


OL805217

## Description

**Warning:** Because this command file runs with superuser authority and its whole purpose is to remove files, it has the potential for unexpected results. Before installing a new **skulker**, test any additions to its file removal criteria by running it manually using the **xargs -p** command. After you have verified that the new **skulker** removes only the files you want removed, you can install it.

The **skulker** command is a shell command file for periodically purging obsolete or unneeded files from file systems. Candidate files include those in **/tmp**, **.bak** files older than a specified age, and files named **a.out**, **core**, or **ed.hup**.

The **skulker** command is normally invoked daily, often as part of an accounting procedure run by **cron** during off-peak periods. Individual sites should modify **skulker** to suit local needs following the patterns shown in the distributed version. Local users should be made aware of the criteria for automatic file removal.

The **find** and **xargs** commands form a powerful combination for use in **skulker**. Most file selection criteria can be expressed conveniently with **find** expressions. The resulting file list, generated with the **-print** flag of the **find** command, can then be segmented and inserted into **rm** commands using **xargs** to reduce the overhead that would result if each file were deleted with a separate command.

## Related Information

The following commands: "**cron**" on page 220, "**find**" on page 422, "**rm**" on page 833, and "**xargs**" on page 1232.

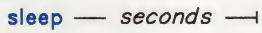
The section on using the **skullker** in *Managing the AIX Operating System.*

# sleep

## Purpose

Suspends execution for an interval.

## Syntax




OL805218

## Description

The **sleep** command suspends execution of a process for the interval specified by *seconds*. *seconds* can range from 1 to 65,536 seconds.

## Examples

1. To run a command after a certain amount of time has passed:

```
echo  "SYSTEM SHUTDOWN IN 10 MINUTES!" | wall
(sleep 300; echo "SYSTEM SHUTDOWN IN 5 MINUTES!" | wall) &
(sleep 540; echo "SYSTEM SHUTDOWN IN 1 MINUTE!" | wall) &
(sleep 600; shutdown)  &
```

   This command sequence warns all users 10 minutes, 5 minutes, and 1 minute before the system is shut down.

2. To run a command at regular intervals:

```
while true
do
   date
   sleep 60
done
```

   This shell procedure displays the date and time once a minute. To stop it, press INTERRUPT (**Alt-Pause**).

## Related Information

The following commands: "**shutdown**" on page 946 and "**wall**" on page 1208.

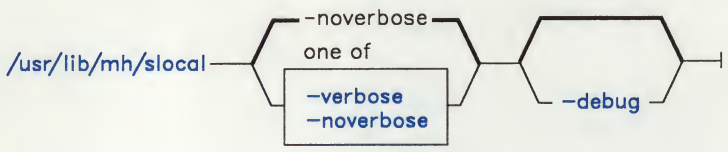
The **alarm** system call and the **sleep**, **pause**, and **signal** subroutines in *AIX Operating System Technical Reference*.

# slocal

## Purpose

Processes incoming mail.

## Syntax





AJ2FL259

## Description

The **slocal** command is used to perform a set of actions each time a message is sent to the user. **slocal** is not designed to be run directly by the user; it is designed to be called by the **sendmail** command. The **slocal** command is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **sendmail** command invokes the **slocal** command when it encounters the following line in **$HOME/.forward**:

```
| /usr/lib/mh/slocal
```

For each new incoming message, **slocal** performs the actions specified in the **.maildelivery** file. If **slocal** cannot find the file **$HOME/.maildelivery**, **slocal** uses the default file **/usr/lib/mh/maildelivery**. If the delivery request fails, **slocal** delivers the message to **/usr/mail/$USER**.

The actions that can be specified in **.maildelivery** are described in *AIX Operating System Technical Reference* under **mhooks**.

## Flags

| | |
|---|---|
| **-debug** | Provides information for debugging. |
| **-help** | Displays help information for the command. |

| | |
|---|---|
| **-noverbose** | Does not display information as the system executes commands in the maildelivery file. This flag is the default. |
| **-verbose** | Displays information as the system executes commands in the maildelivery file. |

## Files

| | |
|---|---|
| /usr/lib/mh/mtstailor | The MH tailor file. |
| /usr/lib/mh/maildelivery | The default MH local delivery instructions file. |
| $HOME/.maildelivery | The user's local delivery instructions file. |
| $HOME/.forward | The user's default message filter file. |

## Related Information

The following commands: "**rcvdist**" on page 808, "**rcvpack**" on page 810, "**rcvstore**" on page 812, "**rcvtty**" on page 815, "**sendmail**" on page 897.

The **mh-format**, **mhook**, **mh-mail**, and **mh-profile** file formats in *AIX Operating System Technical Reference.*

"Overview of the Message Handling Package" in *Managing the AIX Operating System.*

# sno

## Purpose

Provides a SNOBOL interpreter.

## Syntax


OL805219

## Description

The **sno** command provides a SNOBOL compiler and interpreter, with some differences from standard SNOBOL. It reads the named *file*s and the standard input. It compiles all input through a statement containing the label **end**. The rest is available to **syspit**. The **sno** command differs from SNOBOL in the following ways:

- There are no unanchored searches. To get the same effect:

```
a ** b              Unanchored search for b
a *x* b = x c       Unanchored assignment.
```

- There is no back referencing.

```
x = "abc"
a *x* x             Unanchored search for abc.
```

- Function declaration is done at compile time by the use of the (nonunique) label **define**. Execution of a function call begins at the statement following the **define**. Functions cannot be defined at run time, and the use of the name **define** is pre-empted. There is no provision for automatic variables other than parameters. Examples:

```
define f()
define f(a, b, c)
```

- All labels except **define** (even **end**), must have a nonempty statement.
- Labels, functions, and variables must all have distinct names. In particular, the nonempty statement on **end** cannot merely name a label.
- If **start** is a label in the program, program execution begins there. If not, execution begins with the first executable statement. **define** is not an executable statement.
- There are no built-in functions.

- Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators \ (backslash) and * (asterisk) must be set off by spaces.
- The right side of assignments must be nonempty.
- Either ' (single quotation mark) or " (double quotation mark) can be used for literal quotation marks.
- The pseudo-variable **sysppt** is not available.

## Related Information

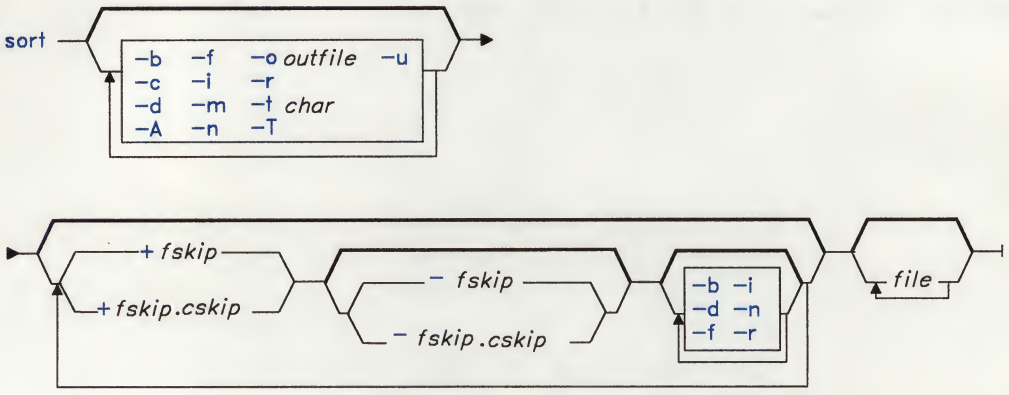
The following command: "**awk**" on page 81.

# sort

## Purpose

Sorts or merges files.

## Syntax




OL805380

## Description

The **sort** command sorts lines in its input *file*s and writes the result to standard output. It treats all of its input *file*s as one file when it performs the sort. A - (minus) in place of a file name specifies standard input. If you do not specify any file names, it sorts standard input.

The default sort key (the part of the line used for sorting) is an entire line. Default ordering is lexicographic by characters in the collating sequence. The file **/usr/pub/ascii** shows the default collating sequence. To change the default collating sequence, see "**ctab**" on page 257.

The two numbers, *fskip* and *cskip*, specify the sort key. Both numbers have two parts, as follows:

*+fskip.cskip*
*-fskip.cskip*

The *fskip* specifies the number of fields to skip from the beginning of the input line, and *cskip* specifies the number of additional characters to skip to the right beyond that point. For both the starting point (*+fskip.cskip*) and the ending point (*-fskip.cskip*) of a sort key, *fskip* is measured from the beginning of the input line, and *cskip* is measured from the last field skipped. If you omit *.cskip*, .0 is assumed. If you omit *fskip*, 0 is assumed. If you omit the ending field specifier (*-fskip.cskip*), the end of the line is the end of the sort key.

You can supply more than one sort key by repeating *+fskip.cskip* and *-fskip.cskip*. In cases where you specify more than one sort key, keys specified further to the right on the command line are compared only after all earlier keys are sorted. For example, if the first key is to be sorted in numerical order and the second in dictionary order, all strings that start with the number one are sorted alphabetically before the strings that start with the number two. Lines that are identical in all keys are sorted with all characters significant. You can also specify different flags for different sort keys in multiple sort keys. See the examples for illustration.

A field is one or more characters bounded by the beginning of a line and the current field separator, or one or more characters bounded by a the field separator on either side. The space character is the default field separator.

**Notes:**

1. Lines longer than 1024 are truncated.
2. The maximum number of fields on a line is 10.
3. The **sort** command will not process files containing imbedded commands.

## Flags

**-A** Sorts on a byte-by-byte basis. This sort is functionally compatible with the Version 1.1 **sort** command, prior to the addition of international character support.

**-b** Ignores leading blanks, spaces, and tabs in sort key comparisons.

**-c** Checks that the input is sorted according to the ordering rules specified in the flags. Displays nothing unless the file is not sorted.

**-d** Sorts in dictionary order. Only letters, digits and blanks are considered in comparisons.

**-f** Merges uppercase and lowercase letters. Case is not considered in the sorting, so that initial-capital words and all-capital words are not grouped together at the beginning of the output.

**-i** Sorts only by characters in the ASCII range octal 040-0176 (all printable characters and the space character) in non-numeric comparisons.

**Japanese Language Support Information**

Sorts only by printable characters in non-numeric comparisons.

**-m** Merges only; the input is already sorted.

**-n** Sorts any initial numeric strings (consisting of optional blanks, optional minus signs, and zero or more digits with optional decimal point) by arithmetic value. The **-n** flag automatically gives you the **-b** flag.

**-o** *outfile* Directs output to *outfile* instead of standard output. *outfile* can be the same as one of the input *files*.

**-r** Reverses the order of the specified sort.

**-t***char* Sets field separator character to *char*. To specify the tab character as the field separator, you must enclose it in single quotation marks (' ').

**-T** Uses current directory instead of default directory for temporary files.

**-u** Suppresses all but one in each set of equal lines. Ignored characters (such as leading tabs and spaces) and characters outside of sort keys are not considered in this type of comparison.

## Examples

1. To perform a simple sort:

```
sort  fruits
```

This displays the contents of `fruits` sorted in ascending lexicographic order. This means that the characters in each column are compared one by one, including spaces, digits, and special characters. For instance, if `fruits` contains the text:

```
banana
orange
Persimmon
apple
%%banana
apple
ORANGE
```

then **sort** displays:

```
%%banana
ORANGE
Persimmon
apple
apple
banana
orange
```

This order follows from the fact that in the ASCII collating sequence, % (percent sign) precedes the uppercase letters, which precede the lowercase letters. If the system uses a character set other than ASCII, your results may be different.

2. To sort in dictionary order:

```
sort  -d  fruits
```

This sorts and displays the contents of `fruits`, comparing only letters, digits, and blanks. If `fruits` is the same as in Example 1, then **sort** displays:

```
ORANGE
Persimmon
apple
apple
%%banana
banana
orange
```

The `-d` flag tells **sort** to ignore the % character because it is not a letter, digit, or blank. This puts `%%banana` next to `banana`.

3. To group lines that contain uppercase and special characters with similar lowercase lines:

```
sort  -d  -f  fruits
```

This ignores special characters (`-d`) and differences in case (`-f`). Given the `fruits` of Example 1, this displays:

```
apple
apple
%%banana
banana
ORANGE
orange
Persimmon
```

4. To sort as in Example 3 and remove duplicate lines:

```
sort  -d  -f  -u  fruits
```

The -u flag tells **sort** to remove duplicate lines, making each line of the file unique. This displays:

```
apple
%%banana
orange
Persimmon
```

Note that not only was the duplicate apple removed, but banana and ORANGE as well. These were removed because the -d told **sort** to treat %%banana as if it were banana, and the -f told it to treat ORANGE as orange. Thus, **sort** considered %%banana to be a duplicate of banana and ORANGE a duplicate of orange.

**Note:** There is no way to predict which duplicate lines sort -u will keep and which it will remove.

5. To sort as in Example 3 and remove duplicates, unless capitalized or punctuated differently:

```
sort  -u  +0  -d  -f  +0  fruits
```

The +0 -d -f does the same type of sort done with -d -f in Example 3. Then the +0 performs another comparison to distinguish lines that are not actually identical. This prevents -u from removing them.

Given the fruits file shown in Example 1, the added +0 distinguishes %%banana from banana and ORANGE from orange. However, the two instances of apple are identical, so one of them is deleted.

```
apple
%%banana
banana
ORANGE
orange
Persimmon
```

6. To specify the character that separates fields:

```
sort  -t:  +1  vegetables
```

This sorts vegetables, comparing the text that follows the first colon on each line. The +1 tells **sort** to ignore the first field and to compare from the start of the second field to the end of the line. The -t: tells **sort** that colons separate fields. If vegetables contains:

```
yams:104
turnips:8
potatoes:15
carrots:104
green beans:32
radishes:5
lettuce:15
```

then **sort** displays:

```
carrots:104
yams:104
lettuce:15
potatoes:15
green beans:32
radishes:5
turnips:8
```

Note that the numbers are not in numeric order. This happened because a lexicographic sort compares each character from left to right. In other words, "3" comes before "5" and "2" comes before " ", so "32" comes before "5 ".

7. To sort numbers:

```
sort  -t: +1  -n  vegetables
```

This sorts `vegetables` numerically on the second field. If `vegetables` is the same as in Example 6, then **sort** displays:

```
radishes:5
turnips:8
lettuce:15
potatoes:15
green beans:32
carrots:104
yams:104
```

8. To sort on more than one field:

```
sort  -t: +1  -2  -n  +0  -1  -r  vegetables
```

This performs a numeric sort on the second field (`+1 -2 -n`). Within that ordering, it sorts the first field in reverse alphabetic order (`+0 -1 -r`). The output looks like this:

```
radishes:5
turnips:8
potatoes:15
lettuce:15
green beans:32
yams:104
carrots:104
```

Now the lines are sorted in numeric order. When two lines have the same number, they appear in reverse alphabetic order.

9. To replace the original file with the sorted text:

```
sort  -o  vegetables  vegetables
```

This stores the sorted output into the file `vegetables` (`-o vegetables`).

## Files

| | |
|---|---|
| sort.c | Contains sort definitions. |

## Related Information

The following commands: "**comm**" on page 183, "**join**" on page 547, and "**uniq**" on page 1118.

The "Overview of International Character Support" in *Managing the AIX Operating System.*

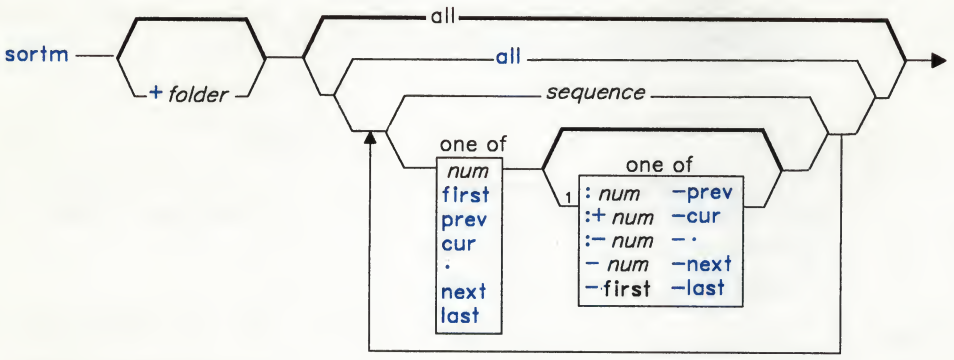
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

# sortm

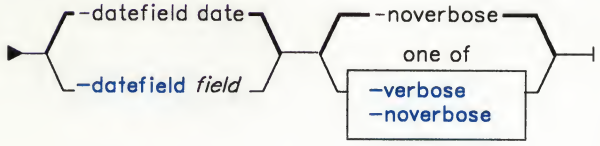
## Purpose

Sorts messages.

## Syntax




AJ2FL209





AJ2FL211

[1] Do not put a blank between these items.

OL805308

## Description

The **sortm** command is used to sort messages. The **sortm** command is part of the MH (Message Handling) package and can be used with MH and AIX commands.

The **sortm** command sorts the messages according to the date in a header field. By default, **sortm** parses the **Date:**, but you can can use the **-datefield** flag to specify another field. **sortm** numbers the sorted messages consecutively beginning with 1 (one). Messages that are in the folder, but not specified to be sorted, are placed after the sorted messages. **sortm** displays a message if it cannot parse a date field.

## Flags

**-datefield** *field* Specifies the header field to be used in the sort. The default field is **Date:**.

+*folder msgs* Specifies the messages to be sorted. You can use use the following message references when specifying *msgs*:

| | | |
|---|---|---|
| *num* | **first** | **prev** |
| **cur** | **.** | **next** |
| **last** | **all** | *sequence* |

The default is all messages in the current folder. If you specify a folder, it becomes the current folder. The current message remains the current message, even if it moves during the sort.

**-help** Displays help information for the command.

**-noverbose** Does not display information during the sort. This flag is the default.

**-verbose** Displays information during the sort. This information allows you to monitor the steps involved.

## Profile Entries

**Current-Folder:** Sets your default current folder.
**Path:** Specifies your *user_mh_directory*.

## Files

$HOME/.mh_profile The MH user profile.

## Related Information

The MH command "**folder**" on page 429.

The **mh-profile** file in *AIX Operating System Technical Reference*.

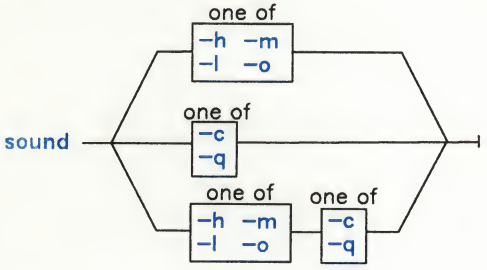
The "Overview of the Message Handling Package" in *Managing the AIX Operating System*.

# sound

## Purpose

Controls the volume and click of the keyboard speaker.

## Syntax




OL805415

## Description

The **sound** command controls the volume of the sound output (the console bell and the keyboard click) and, additionally, whether or not the keyboard click is produced. You can modify these two sound characteristics independently of each other.

The system startup process sets the sound volume to medium.

**Note:** You can run **sound** only from the console (**/dev/console**).

## Flags

You must select at least one flag from the following two groups of flags or, optionally, one flag from each of the two groups. The first group of flags controls the volume of all sound output:

**-h** Sets the volume to high.

**-l** Sets the volume to low.

**-m** Sets the volume to medium.

**-o** Turns the volume off.

The second group of flags controls whether or not click sounds are produced:

**-c** Turns clicking on.

**-q** Turns clicking off (quiet).

## Example

To set the volume to low and turn the click function on:

```
sound -lc
```

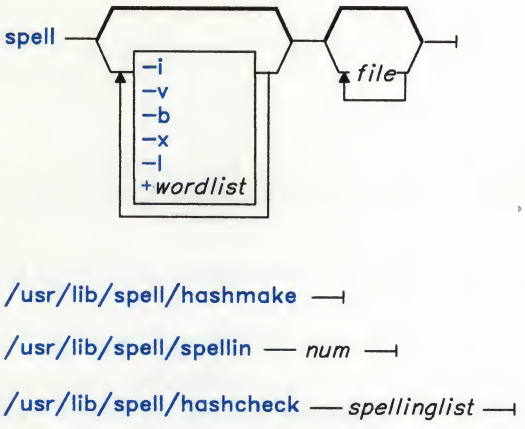
In addition to turning on the keyboard click (`-c`), this command sets the volume of both the bell and the click to low (`-l`).

# spell

## Purpose

Finds spelling errors.

## Syntax




OL805304

## Description

The **spell** command reads words in *file* and compares them to those in a spelling list. Words that cannot be matched in the spelling list or derived from words in the spelling list (by applying certain inflections, prefixes, and/or suffixes) are written to standard output. If you do not specify a file to read, **spell** reads standard input.

The **spell** command ignores the same **troff**, **tbl**, and **eqn** constructs as the **deroff** command.

The coverage of the spelling list is uneven. You should create your own dictionary of special words used in your files.

Certain auxiliary files can be specified by file name parameters; see "Files" on page 971. Copies of all output are accumulated in the history file.

Three routines help maintain and check the hash lists used by **spell**.

| | |
|---|---|
| **/usr/lib/spell/hashmake** | Reads a list of words from standard input and writes the corresponding nine-digit hash code to standard output. |
| **/usr/lib/spell/spellin** *num* | Reads *num* hash codes from standard input and writes a compressed spelling list to standard output. |
| **/usr/lib/spell/hashcheck** *spellinglist* | Reads a compressed *spellinglist* and recreates the nine-digit hash codes for all the words in it; it writes these codes to standard output. |

## Flags

| | |
|---|---|
| **-b** | Checks British spelling. |
| **-i** | Suppresses processing of included files. |
| **-l** | Follows the chain of all included files (**.so** and **.nx** formatting commands). Without this flag, **spell** follows chains of all included files except for those beginning with **/usr/lib**. |
| **-v** | Displays all words not literally in the spelling list and indicates plausible derivations from the words. |
| **-x** | Displays every plausible word stem with an = (equal sign). |
| +*wordlist* | Checks *wordlist* for additional word spellings. *wordlist* is the name of a file you provide that contains a sorted list of words, one per line. With this flag, you can specify a set of correctly spelled words (in addition to **spell**'s own spelling list) for each job. |

## Examples

1. To check your spelling:

```
spell  chap1  >mistakes
```

   This creates a file named `mistakes` containing all the words found in `chap1` that are not in the system spelling dictionary. Some of these may be correctly spelled words that **spell** does not know about. It is a good idea to save the output of **spell** in a file because the word list may be long.

2. To check British spelling:

```
spell  -b  chap1  >mistakes
```

   This checks `chap1` against the British dictionary and writes the questionable words in `mistakes`.

3. To see how **spell** derives words:

```
spell  -v  chap1  >deriv
```

This lists the words that are not found literally in the dictionary, but are derived forms of dictionary words. The prefixes and suffixes used to form the derivative are indicated for each word. Words that do not appear in the dictionary at all are also listed.

4. To check your spelling against an additional word list:

```
spell  +newwords  chap1
```

This checks the spelling of words in `chap1` against the system dictionary and against `newwords`. The file `newwords` lists words in alphabetical order, one per line. You can create this file with a text editor, such as **ed**, and alphabetize it with the **sort** command.

## Files

| | |
|---|---|
| D_SPELL=/usr/lib/spell/hlist[ab] | Hashed spelling lists, American and British. |
| S_SPELL=/usr/lib/spell/hstop | Hashed stop list. |
| H_SPELL=/usr/lib/spell/spellhist | History file. |
| /usr/lib/compress | Executable shell program to compress the history file. |
| /usr/lib/spell/spellprog | Program. |

## Related Information

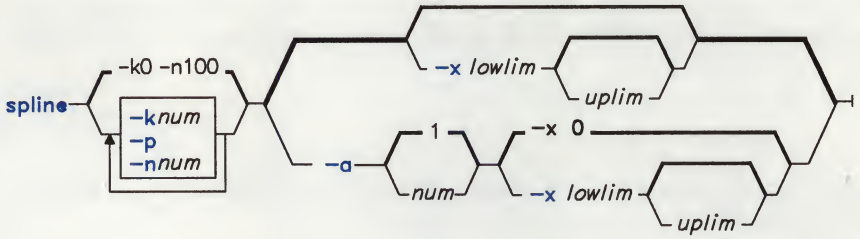
The following commands: "**deroff**" on page 313, "**eqn**, **neqn**, **checkeq**" on page 395, "**sed**" on page 887, "**sort**" on page 958, "**tbl**" on page 1053, "**tee**" on page 1060, and "**troff**" on page 710.

# spline

## Purpose

Interpolates smooth curve.

## Syntax




OL805261

## Description

The **spline** command reads from the standard input pairs of numbers that represent the coordinates of a point on an x,y axis. From this input, **spline** calculates the coordinates of points to form a smooth curve through the points in the input set. It then writes these points to standard output. The output points are approximately equally spaced and includes the points that you provided as input. The cubic spline output has two continuous derivatives, and enough points so that when plotted with the **graph** command it looks smooth.

When data is not strictly monotone in *x*, **spline** reproduces the input without interpolating extra points.

You can only use 1,000 input points.

## Flags

**-a***num* Supplies abscissas automatically; spacing is given by the next parameter or is assumed to be 1 if the next parameter is not a number.

**-k***num* Uses the constant *num* in the boundary value calculation:

$$y_0 = ky_1 , y = numy$$

The default for *num* is zero.

| | |
|---|---|
| **-n***num* | Spaces output points so that approximately *num* intervals occur between the lower and upper *x* limits (set with the **-x** flag). The default *num* is 100. |
| **-p** | Makes output periodic, that is, matches derivatives at ends. First and last input values should normally agree. |
| **-x***lowlim*[*uplim*] | Sets lower and upper *x* limits as *lowlim* and *uplim*. Normally, these limits are calculated from the data. Automatic abscissas start at lower limit, defaults to zero. |

## Related Information

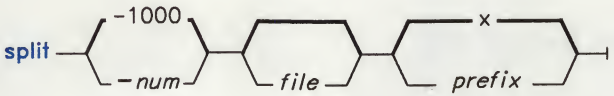
The following command: "**graph**" on page 494.

# split

## Purpose

Splits a file into pieces.

## Syntax




OL805262

## Description

The **split** command reads *file* and writes it in *num*-line pieces (default 1000 lines) to a set of output files. The name of the first output file is *prefix***aa**, the second is *prefix***ab**, and so on lexicographically, through *prefix***zz** (a maximum of 676 files). *prefix* cannot be longer than 12 characters. If you do not specify an output name, **x** is assumed.

If you do not specify an input file, or if you specify - (minus) in place of *file*, then **split** reads standard input.

## Examples

1. To split a file into 1000-line segments:

   ```
   split  book
   ```

   This splits `book` into 1000-line segments named **xaa**, **xab**, **xac**, and so forth.

2. To split a file into 50-line segments and specify the file name prefix:

   ```
   split  -50  book  sect
   ```

   This splits `book` into 50-line segments named `sect`**aa**, `sect`**ab**, `sect`**ac**, and so forth.

## Related Information

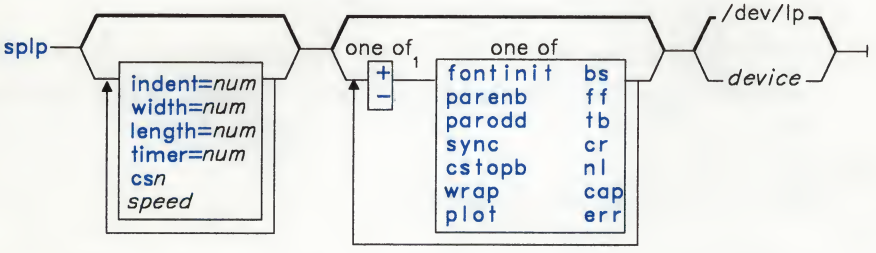
The following commands: "**bfs**" on page 110 and "**csplit**" on page 252.

# splp

## Purpose

Changes or displays printer driver settings.

## Syntax




OL805263

[1] Do not put a blank between these items.

OL805308

## Description

The **splp** command changes or displays settings for a printer driver (*device*). The default *device* is **/dev/lp0**. If you do not specify any flags, **splp** reports the current settings for the specified *device*. Select flags to change the current settings. No other processing is done, and there is no other output.

The changes that **splp** makes remain in effect until the next time you restart the system or rerun **splp**. You can run **splp** from the **/etc/rc** command file to configure your printer each time you start up the system.

**Note:** When the **print** command is used with the backend **piobe**, **splp** is set as **+plot**. Any parameters set by the user are ignored. If a file is redirected using the **cat** command instead of the **print** command, the **splp** settings are active.

### Japanese Language Support Information

This command has not been modified to support Japanese characters.

## Flags

**+ascii )-ascii(**
Selects ASCII )**PostScript**( mode for the Personal Proprinter Adapter.

**timer** = *num* Sets the time out period to *num* seconds, where *num* is an integer.

**indent** = *num* Indents *num* columns, where *num* is an integer.

**length** = *num* Prints *num* lines per page, where *num* is an integer.

**width** = *num* Prints *num* columns, where *num* is an integer

**+bs (-bs)** Sends (does not send) backspaces to the printer.

**+cap (-cap)** Converts (does not convert) all lowercase characters to uppercase.

**+cr (-cr)** Sends carriage returns (translates carriage returns to carriage return - line feeds).

**+cstopb (-cstopb)**
Selects 2 (1) stop bits per character.

**cs5 cs6 cs7 cs8**
Selects character size. See **termio** in *AIX Operating System Technical Reference* for additional information on character size.

**+err (-err)** Issues (does not issue) a signal (SIGIOINT) upon receiving a VRM error and attempts to resume I/O.

**+ff (-ff)** Sends form feeds (simulates a form feed with line feeds or carriage returns).

**+fontinit (-fontinit)**
Indicates that fonts are (are not) loaded. Use this flag to control the initialization of fonts for the 3812 Pageprinter or the Personal Proprinter Adapter.

**+nl (-nl)** Sends line feeds (translates line feeds to line feed - carriage returns).

**+parenb (-parenb)**
Enables (disables) parity generation and detection.

**+parodd (-parodd)**
Selects odd (even) parity.

**+plot** Sends all characters to the printer unmodified. This overrides other settings.

**-plot** Translates characters according to the settings.

**+sync (-sync)**
Does not (does) return immediately without waiting for all data to be sent out.

**+tb (-tb)** Expands (does not expand) tabs on eight position boundaries.

**+wrap (-wrap)**
Wraps (truncates) characters beyond the specified **width** to the next line and (with **+wrap**), prints " . . . " before the new-line character.

**50 75 110 134 150 300 600 1200 1800 2400 4800 9600 exta extb**
Sets the *speed* to the specified number of bits per second (**exta** is 19200).

## Examples

1. To display the current printer settings:

   ```
   splp
   ```

2. To change the printer settings:

   ```
   splp  width=80  +wrap  +cap
   ```

   This changes the settings of the **/dev/lp** printer for 80-column paper (**width**=80). It wraps each line that is more than 80 columns wide onto a second line (**+wrap**), and prints all alphabetic characters in uppercase (**+cap**).

## Related Information

The following command: "**lp**" on page 593.

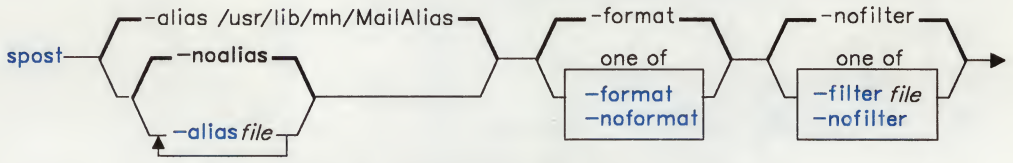
The **lp** file in *AIX Operating System Technical Reference.*

# spost

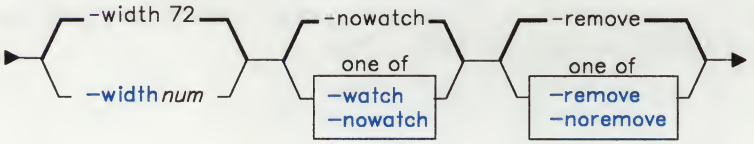
## Purpose

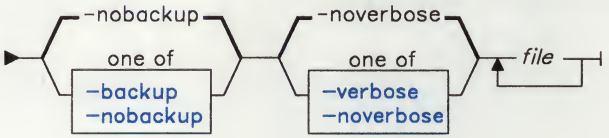
Delivers a message.

## Syntax




AJ2FL251





AJ2FL252

## Description

The **spost** command is used to route messages to the proper destinations. **spost** is not designed to be run directly by the user; it is designed to be called by other programs. The **spost** command is part of the MH (Message Handling) package.

The **spost** command searches all components of a message that specify a recipient's address and parses each address to check for proper format. **spost** puts proper addresses in the standard format and invokes the **sendmail** command. **spost** perform a similar function as the **post** command, but **spost** does less internal error checking than **post**.

## Flags

**-alias Files** Searches the specified mail alias file for addresses. You can use this flag repetitively to specify multiple mail alias files. **spost** automatically searches the file **/usr/lib/mh/MailAliases**.

**-backup** Renames the message file by placing a **,** (comma) before the file name after **spost** successfully posts the message.

**-filter Files** Uses the header components in the specified file for copies of the message sent to **Bcc:** recipients.

**-format** Puts all recipient addresses in a standard format for the delivery transport system. This flag is the default.

**-help** Displays help information for the command.

**-nofilter** Strips the **Bcc:** header from the message and sends it to recipients specified in the **Bcc:** component. This flag is the default.

**-noalias** Does not use any alias files for delivering the message.

**-nobackup** Does not rename the message after posting the file. This flag is the default.

**-noformat** Does not alter the format of the recipient addresses.

**-noremove** Does not remove the temporary message file after posting the message.

**-noverbose** Does not display information during the delivery of the message to the **sendmail** command. This flag is the default.

**-nowatch** Does not display information during delivery by the **sendmail** command. This flag is the default.

**-remove** Removes the temporary message file after the successful completion of posting the message. This flag is the default.

**-verbose** Displays information during the delivery of the message to the **sendmail** command. This information allows you to monitor the steps involved.

**-watch** Displays information during the delivery of the message by the **sendmail** command. This information allows you to monitor the steps involved.

**-width** *num* Sets the width of components that contain addresses. The default is 72 columns.

## Files

| | |
|---|---|
| $HOME/.mh_profile | The MH user profile. |
| /temp/pst*num* | The temporary message file. |

## Related Information

Other commands: "**ali**" on page 48, "**conflict**" on page 196, "**mhmail**" on page 646, "**post**" on page 758, "**send**" on page 893, "**sendmail**" on page 897, and "**whom**" on page 1222.

The **mh-alias**, **mh-format**, **mh-mail**, and **mh-profile** files in *AIX Operating System Technical Reference*.

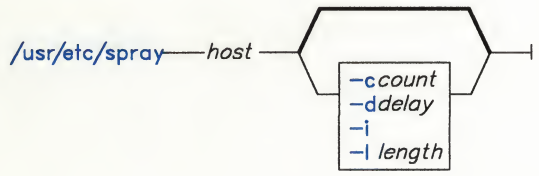
"Overview of the Message Handling Package" in *Managing the AIX Operating System*.

# spray

## Purpose

Sends specified number of packets to host when NFS is installed.

## Syntax




OL805509

## Description

The **spray** command sends a one-way stream of packets to *host* using RPC. It reports how many packets were received, and at what transfer rate. The *host* parameter can be a network address or a name.

**Japanese Language Support Information**

If Japanese Language Support is installed on your system, this command is not available.

## Flags

**-c** Specifies the number of packets to send. The default is the number of packets required to make the total stream size 100,000 bytes.

**-d** Specifies the time, in microseconds, the system pauses between sending each packet. The default is 0.

**-i** Specifies ICMP echo packets rather than RPC. Since ICMP echoes automatically, it creates a two-way stream.

**-l** Number of bytes in the Ethernet packet that holds the RPC call message. The default is 86 bytes, which is the length of RPC and UDP/IP headers.

The data is encoded using XDR. Since XDR deals only with 32-bit quantities, **spray** rounds smaller *length* values up to the nearest 32-bit value.

**Note:** If the **length** is greater than 1514, the RPC call does not fit into one Ethernet packet, and the **length** field will not have a simple correspondence to the Ethernet packet size.

## Related Information

The following command: "**sprayd**" on page 983.

# sprayd

## Purpose

Receives packets sent by the **spray** command

## Syntax


OL805510

## Description

The **sprayd** daemon receives the packets sent when the **spray** command is issued. The **inetd** daemon invokes **sprayd**.

**Japanese Language Support Information**

If Japanese Language Support is installed on your system, this command is not available.

## File

/etc/inetd.conf TCP/IP configuration file.

## Related Information

The following command: "**spray**" on page 981.

# stat

## Purpose

Provides tools for analyzing numerical data.

## Description

The **stat** commands, residing in **/usr/bin/graf**, provide a package of tools for analyzing data. All numerical data are stored in vectors. A ***vector*** is a sequence of numbers separated by delimiters, where a number has the form:

[*sign*](*digits*)(*.digits*)[**e**[*sign*]*digits*]

Fields surrounded by brackets are optional; one or both of those surrounded by parentheses are required. Any input character that is not part of a number is assumed to be a delimiter.

Vectors are text strings that can be stored in text files and created and modified by text editors.

**Note:** Some commands limit the size of an input vector.

These commands can be divided into four classes:

- Those that produce an output vector based upon definable parameters (generators).
- Those that operate upon an input vector and output the resulting value (transformers).
- Those that perform mathematical or statistical operations on vectors (summarizers).
- Those that convert vectors into a format that can be viewed pictorially (translators).

The following parameters are used to designate the expected type of the value:

*c* A character value.

*i* An integer value.

*f* A floating-point or integer value.

*file* A file name.

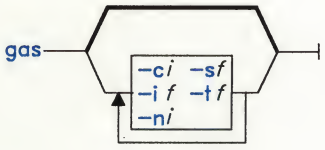
*vector* A vector taken from standard input or the name of a file containing a vector. Except for the **gas**, **prime**, and **rand** commands, all of the commands discussed under **stat** read vectors from standard input (by default) or from text files as specified on the command line. A file name of - (minus) specifies standard input in a file list.

*string* A character string (quoted if it includes white space).

## Commands That Produce Definable Vectors (Generators)

### gas

#### *Syntax*




OL805513

#### *Description*

The **gas** command produces an additive sequence.

#### *Flags*

**-c***i* Specifies the number of columns per line of output (5 by default).

**-i***f* Specifies the increment between successive elements (1 by default).

**-n***i* Specifies the number of elements in the vector (10 by default).

**-s***f* Specifies the starting point of the sequence (1 by default).

**-t***f* Specifies the end of the sequence (infinity by default).

#### *Examples*

1. To generate the numbers 1 through 10:

```
gas
```

2. To generate the sequence .01 .02 .03 .04 .05:

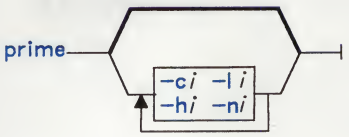
```
gas  -s.01,t.05,i.01
```

3. To generate the sequence 3 5 3 5:

```
gas  -s3,t5,i2,n4
```

## prime

### *Syntax*




OL805514

### *Description*

The **prime** command generates consecutive prime numbers.

### *Flags*

**-c***i* Specifies the number of columns per line of output (5 by default).

**-h***i* Specifies the high boundary (infinity by default).

**-l***i* Specifies the low boundary (2 by default).

**-n***i* Specifies the number of elements in the sequence (10 by default).

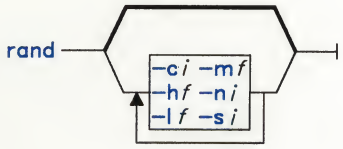
### *Example*

To generate all prime numbers between 200 and 300:

```
prime  -l200,-h300
```

## rand

### *Syntax*




OL805515

### *Description*

The **rand** command generates a random sequence of numbers.

### *Flags*

**-c***i* Specifies the number of columns per line of output (5 by default).

**-h***f* Specifies the high boundary (1 by default).

**-l***f* Specifies the low boundary (0 by default).

**-m***f* Specifies the high boundary where **h***f* = **m***f* + **l***f*.

**-n***i* Specifies the number of elements in the sequence (10 by default).

**-s***i* Specifies a seed number (1 by default).

### *Example*

To produce a random sequence:

```
rand
```

This generates ten random numbers between 0 and 1.

```
rand  -l10,m25,c3
```

This generates ten random numbers between 10 and 35, three per line.

# Commands That Map Input to Output (Transformers)

## abs

### *Syntax*


OL805516

### *Description*

The **abs** command provides the absolute value of a number.

### *Flag*

**-c***i* Specifies the number of columns per line of output (5 by default).

### *Example*

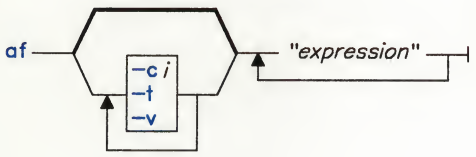
To obtain the absolute value of each element in a vector:

```
abs  -c3  myfile
```

This produces the absolute value of each number in the file `myfile` and displays these values three per line.

## af

### *Syntax*




OL805517

### *Description*

The **af** command performs arithmetic operations on numbers.

### *Expressions*

Expression operands are:

Vectors — File names with the restriction that they must begin with a letter and be composed only of letters, digits, and the _ (underscore) and . (dot) characters. The first unknown file name (one not in the current directory) references standard input.

Functions — The name of a command followed by the command arguments in parentheses. List arguments as you would on the command line.

Constants — Floating-point and integer numbers (but not E notation).

Expression operators are, in order of decreasing precedence:

*'v* — The next value from vector *v*.

*x^y -x* — The value *x* raised to the power *y*; the negation of *x*. Both associate right to left.

*x*y x/y x%y* — The value *x* multiplied by, divided by, modulo *y*, respectively. All associate left to right.

*x+y x-y* — The value *x* plus or minus *y*. Both associate left to right.

*x,y* — The value of *x* followed by the value of *y*. This associates from left to right.

You can use parentheses to alter precedence. Because many of the operator characters are special to the shell, it is good practice to quote expression arguments.

### *Flags*

**-c***i* Specifies the number of columns per line of output (5 by default).

**-t** Cause the output to be titled from the vector on standard input.

**-v** Echoes function expansions.

### *Examples*

1. To perform arithmetic operations:

   ```
   af  "3+4*5"
   ```

   This yields 23.

2. To produce a matrix:

   ```
   af  "A,'A,A+'A,B"
   ```

   This yields a four-column matrix with columns of:

   a. odd elements from vector A

   b. even elements from A

   c. sum of adjacent odd and even elements from A

   d. elements from vector B.

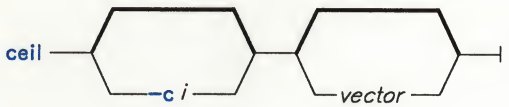
3. To use functions:

   ```
   af  "sin (A)^2"
   ```

   This yields the square of the sin of the elements of vector A.

## ceil

### *Syntax*




OL805518

### *Description*

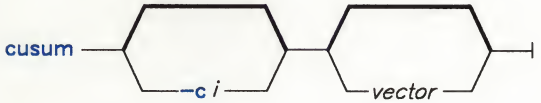
The **ceil** command rounds a number up to the next integer.

### *Flag*

**-c***i* Specifies the number of columns per line of output (5 by default).

## cusum

### *Syntax*




OL805519

### *Description*

The **cusum** command calculates a cumulative sum. Output is a vector with the *i*th element being the sum of the first *i* elements from the input vector.

### *Flag*

**-c***i* Specifies the number of columns per line of output (5 by default).

## exp

### *Syntax*


OL805549

### *Description*

The **exp** command provides the exponential function. Output is a vector with elements e raised to the *x* power, where e is approximately 2.71828 and *x* is each element in the input vector.

### *Flag*

**-c***i* Specifies the number of columns per line of output (5 by default).

## floor

### *Syntax*


OL805550

### *Description*

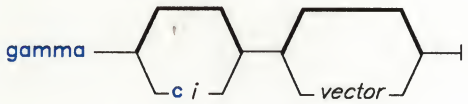
The **floor** command rounds a number down to the nearest integer.

### *Flag*

**-c***i* Specifies the number of columns per line of output (5 by default).

## gamma

### *Syntax*




OL805551

### *Description*

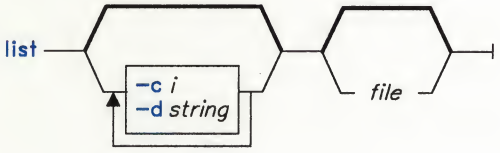
The **gamma** command provides the gamma function.

### *Flag*

**-c***i* Specifies the number of columns per line of output (5 by by default).

## list

### *Syntax*




OL805552

### *Description*

The **list** command lists vector elements.

### *Flags*

**-c***i* Specifies the number of columns per line of output (5 by default).

**-d***string* Specifies delimiter characters. If you do not specify **-d**, any character that is not part of a number is considered a delimiter. If you specify **-d**, the space, tab, and new-line characters, plus the characters in *string* are delimiters.

Only numbers surrounded by delimiters are listed.

### *Examples*

1. To output each element:

```
list  -c3  myfile
```

   This outputs each element in `myfile`, three per line.

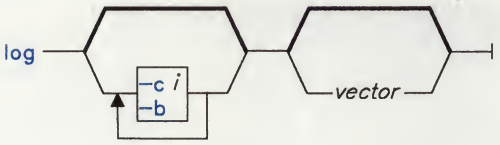
2. To specify delimiters:

```
list  -d\\,  myfile
```

   This outputs each element of `myfile` that is delimited by commas or white space, five per line. A comma requires two backslashes because it is a special character for **list**.

## log

### *Syntax*




OL805520

### *Description*

The **log** command provides the logarithmic function.

### *Flag*

**-c***i* Specifies the number of columns per line of output (5 by default).

**-b***f* Specifies the base (e by default).

### *Example*

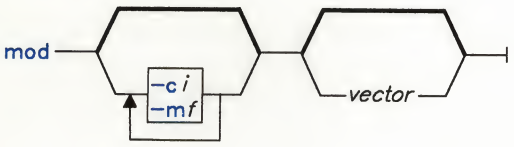
To calculate a logarithm:

```
log  -b2,c3  mydata
```

This outputs the logarithm base 2 of each element in `mydata`, three per line.

## mod

### *Syntax*




OL805521

### *Description*

The **mod** command returns the modulo. The output is a vector with each element being the remainder of dividing the corresponding element from the input vector by the modulus.

**-c***i* Specifies the number of columns per line of output (5 by default).

**-m***f* Specifies the modulus (2 by default).

### *Example*

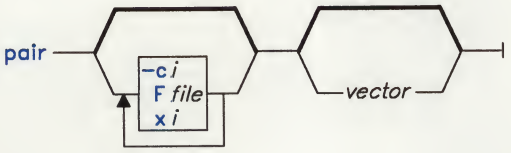
To output remainders:

```
mod  -m8,c3  mydata
```

This outputs the elements of `mydata` modulo 8, three per line.

## pair

### *Syntax*




OL805522

### *Description*

The **pair** command pairs elements. Output is a vector with elements taken alternatively from a base vector and from another input vector.

### *Flags*

**-c***i* Specifies the number of columns per line of output (5 by default).

**-F***file* The file containing the base vector. If you do not specify **-F**, then the base vector comes from standard input. If both the base vector and the paired vector come from standard input, the base vector precedes the paired vector.

**-x***i* The number of elements per group in the base vector (1 by default).

### *Example*

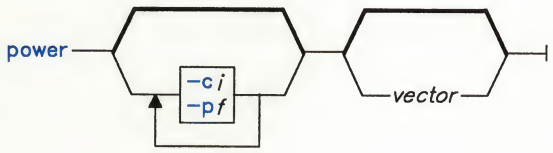
To pair elements:

```
pair  -x3,Fbasefile  datafile
```

This outputs a vector with three elements from `basefile`, one from `datafile`, three from `basefile`, one from `datafile`, and so on.

## power

### *Syntax*




OL805523

### *Description*

The **power** command raises a number to a power.

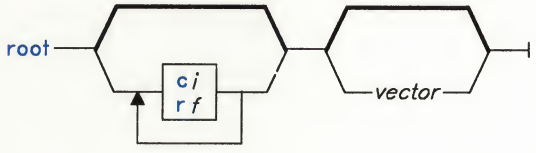
### *Flag*

**-c***i* Specifies the number of columns per line of output (5 by default).

**-p***f* Specifies the power (2 by default).

## root

### *Syntax*




OL805524

### *Description*

The **root** command takes the root of a number.

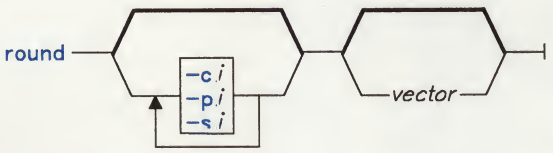
### *Flags*

**-c***i* Specifies the number of columns per line of output (5 by default).

**-r***f* Specifies the root (2 by default).

## round

### *Syntax*




OL805525

### *Description*

The **round** command rounds a number to the nearest integer (.5 rounds to 1).

### *Flags*

**-c***i* Specifies the number of columns per line of output (5 by default).

**-p***i* Specifies the number of places after the decimal point (0 by default).

**-s***i* Specifies the number of significant digits.

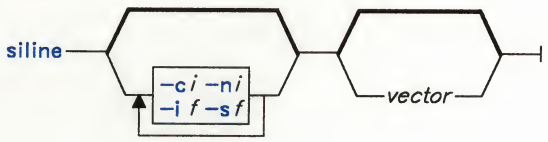
### *Example*

To round numbers to two significant digits:

```
round  -s2,c3  mydata
```

## siline

### *Syntax*




OL805526

### *Description*

The **siline** command generates a line, given slope and intercept.

### *Flags*

**-c***i* Specifies the number of columns per line of output (5 by default).

**-i***f* Specifies the intercept (0 by default).

**-n***i* Specifies the number of positive integers.

**-s***f* Specifies the slope of the line.

### *Example*

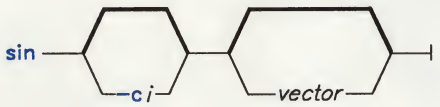
To output a linear fit:

```
siline  -`lref  -o,FA  B`  A
```

This outputs a simple linear fit of vector B on vector A (The **o** flag of **lreg** outputs the slope and intercept in option form of B regressed on A.)

## sin

### *Syntax*




OL805527

### *Description*

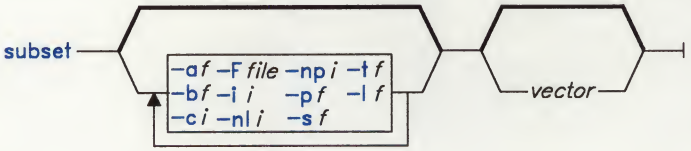
The **sin** command provides the sine function.

### *Flags*

**-c***i* Specifies the number of columns per line of output (5 by default).

## subset

### *Syntax*




OL805528

### *Description*

The **subset** command produces a subset of the numbers in a vector.

### *Flags*

**-a***f* Specifies the number above which subset members are selected.

**-b***f* Specifies the number below which subset members are selected.

**-c***i* Specifies the number of columns per line of output (5 by default).

**-F***file* Specifies the file containing the master vector.

**-i***i* Specifies the increment between successive elements (1 by default).

**-l***f* The number of elements to leave.

**-nl***i* Specifies that the master file contains element numbers to leave.

**-np***i* Specifies that the master file contains element numbers to pick.

**-p***f* The number of elements to pick.

**-s***f* Specifies the starting point of the sequence (1 by default).

**-t***f* Specifies the end of the sequence (32,767 by default).

### *Examples*

1. To specify the even elements of a vector:

```
subset  -i2,s2  myfile
```

2. To specify corresponding elements:

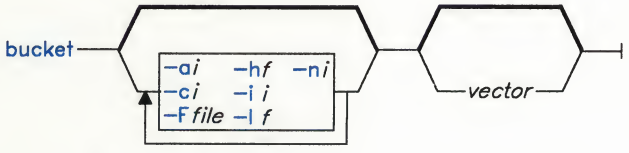
```
subset  -FB,p1  A
```

For each element in B with a 1, output the corresponding element in A.

## Commands That Calculate Statistics (Summarizers)

### bucket

#### *Syntax*




OL805529

#### *Description*

The **bucket** command groups numbers into buckets. The input vector must be sorted. The output vector consists of odd (parenthesized) elements that are bucket limits and even elements that are bucket counts. The count is the number of elements greater than or equal to the lowest limit and less than or equal to the higher limit. Unless otherwise specified, bucket limits are generated based on the input date and the following rule:

#buckets = 1 + log2(#elements)

### *Flags*

**-a***i* Specifies the average size of the bucket.

**-c***i* Specifies the number of columns per line of output (5 by default).

**-F***file* Specifies the file containing bucket boundaries.

**-h***f* Specifies the high boundary (by default, the largest element in the input vector).

**-i***i* Specifies the interval between successive elements.

**-l***i* Specifies the low boundary (by default, the smallest element in the input vector).

**-n***i* Specifies the number of buckets.

### *Example*

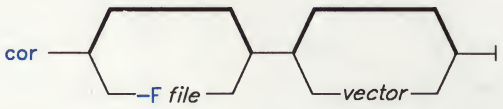
To divide elements into buckets:

```
bucket  -a12,1-5  myfile
```

This outputs limits and counts for the elements of `myfile`, where the lowest limit is `-5` and the average bucket count is `12`.

## cor

### *Syntax*




OL805530

### *Description*

The **cor** command provides the ordinary correlation coefficient. Use the **F** flag to specify the base vector; otherwise it is assumed to come from standard input. Each *vector* is compared to the base vector (both must be of the same length).

### *Flag*

**-F***file* Specifies the file containing base vector.

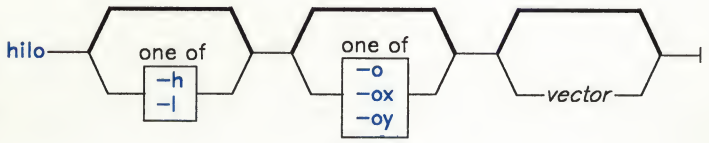
### *Example*

To obtain correlation coefficients:

```
cor  -Ffilea  olddata  newdata
```

This outputs the ordinary correlation coefficients between vectors `filea` and `olddata` and vectors `filea` and `newdata`.

## hilo




OL805531

### *Description*

The **hilo** command finds high and low values across all of the input vectors.

**-h** Finds the high value only.

**-l** Finds the low value only.

**-o** Outputs the high and low values in option form (suitable for **plot**).

**-ox** Outputs the high and low values in option form with **x** prefixed.

**-oy** Outputs the high and low values in option form with **y** prefixed.

### *Example*

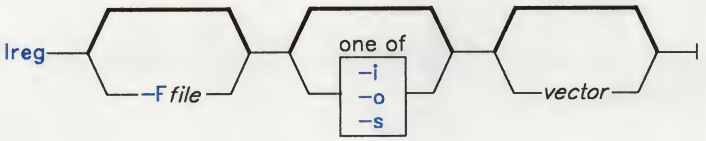
To find the lowest value:

```
hilo  -ox,l file1 file2
```

This finds the lowest value in vectors `file1` and `file2` and outputs it with `xl` prefixed to it.

## lreg

### *Syntax*




OL805532

### *Description*

The **lreg** command provides linear regression. Output is the slope and intercept from a least squares linear regression of each vector on a base vector. The default base vector is the ascending positive integers from zero.

### *Flags*

**-F***file* Specifies a file containing the first vector.

**-i** Outputs only the intercept.

**-o** Outputs the slope and intercepts in option form (suitable for **siline**).

**-s** Outputs only the slope.

### *Example*

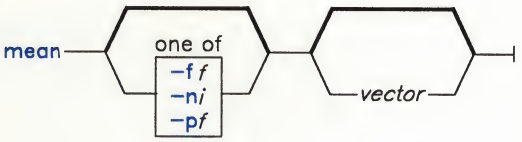
To output only the intercept:

```
lreg  -Fbase,i  mydata
```

This outputs the intercept from the linear regression of vector `mydata` on base vector `base`.

## mean

### *Syntax*




OL805533

### *Description*

The **mean** command calculates the (trimmed) arithmetic mean.

### *Flags*

**-f***f* Specifies the fraction of elements to trim from each end. This is calculated as follows:

$$(1/f)\ k$$

where k is the total number of elements.

**-n***i* Specifies the number of elements to trim from each end.

**-p***f* Specifies the percentage of elements to trim from each end.

### *Example*

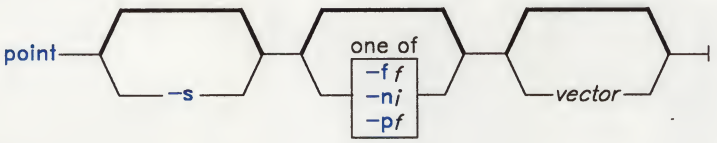
To output the mean:

```
mean  -p.25  mydata
```

This outputs the mean of the middle 50% of the elements of `mydata`; that is, `mydata` is trimmed by 25% of its elements from both ends.

## point

### *Syntax*




OL805534

### *Description*

Output from the **point** command is a linearly interpolated value from the empirical cumulative density function for the input vector. By default, **point** returns the median (50% point).

### *Flags*

**-f***f* Returns the (1/*f*)*100 percent point.

**-n***i* Returns the *i*th element.

**-p***f* Returns the *f**100 percent point.

**-s** Specifies that the input has been sorted.

### *Example*

To output the 25% point:

```
point  -s,p.25  mydata
```

## prod

### *Syntax*

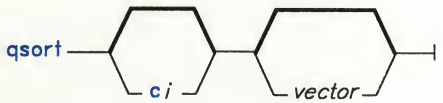

OL805556

### *Description*

The **prod** commands calculates an internal product. Output is the product of the elements in the input vectors.

## qsort

### *Syntax*




OL805536

### *Description*

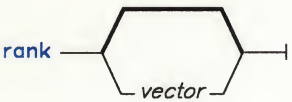
The **qsort** command does a quick sort. Output is a vector of the elements from the input vector in ascending order.

### *Flag*

**-c***i* Specifies the number of columns per line of output (5 by default).

## rank

### *Syntax*



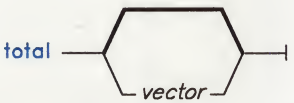

OL805535

### *Description*

The **rank** command ranks vectors. Output is the number of elements in each input vector.

### total

#### *Syntax*



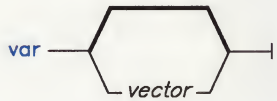

OL805537

#### *Description*

The **total** command calculates a sum total. Output is the sum total of the elements in the input vector(s).

### var

#### *Syntax*




OL805538

#### *Description*

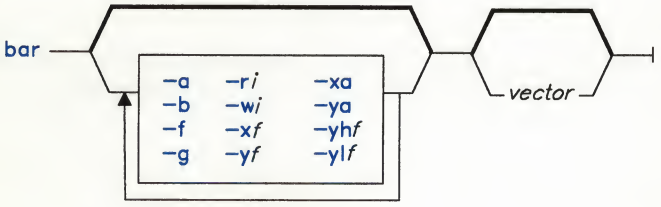
The **var** command calculates the variance.

## Commands That Produce Pictorial Output (Translators)

Input to these commands can be either a vector or a **GPS** object (a format for storing a picture). A picture is defined in a Cartesian plane of 64K points on each axis. The plane, or universe, is divided into 25 square regions numbered 1 to 25 from the lower left to the upper right.

## bar

### *Syntax*




OL805539

### *Description*

The **bar** command builds a bar chart. It operates on an input vector, each element of which defines the height of a bar (y-axis). By default, the x-axis is labeled with positive integers, beginning at 1. For other labels, see **label**.

### *Flags*

**-a** Suppresses the axes.

**-b** Plots the bar chart with bold weight lines (medium is the default weight).

**-f** Does not build a frame around the plot area.

**-g** Suppresses the background grid.

**-r***i* Puts the bar chart in **GPS** region *i*, where *i* is between 1 and 25 inclusive (13 by default).

**-w***i* Specifies the ratio of the bar width to center-to-center spacing expressed as a percentage (50 by default, giving equal bar width and bar space).

**-x***f*
**-y***f* Positions the bar chart in the **GPS** universe with the x-origin (y-origin) at *f*.

**-xa**
**-ya** Does not label the x-axis (y-axis).

**-yh***f* Specifies the y-axis high boundary.

**-yl***f* Specifies the y-axis low boundary.

### *Example*

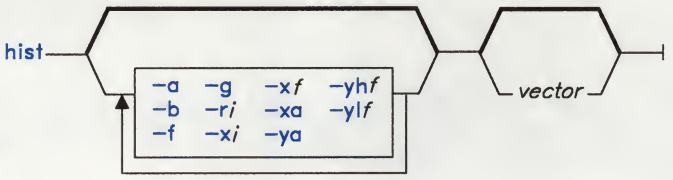
To produce a bar chart:

```
bar  -r10,xa,w75 myfile
```

This outputs the bar chart described by vector `myfile`, located in region 10 of the **GPS** universe, with no x-axis labels. The bar width is 75% of center-to-center spacing.

## hist

### *Syntax*




OL805540

### *Description*

The **hist** command builds a histogram. The input vector is the type produced by **bucket**, of odd rank, with odd elements being limits and even elements being bucket counts.

### *Flags*

**-a** Suppresses axes.

**-b** Plots histogram with bold weight lines (the default weight is medium).

**-f** Does not build a frame around the plot area.

**-g** Suppresses the background grid.

**-r***i* Puts the histogram in **GPS** region *i*, where *i* is between 1 and 25 inclusive (13 by default).

**-x***f*
**-y***f* Positions the histogram in the **GPS** universe with the x-origin (y-origin) at *f*.

**-xa**
**-ya** Does not label the x-axis (y-axis).

**-yh***f* Specifies the y-axis high boundary.

**-yl***f* Specifies the y-axis low boundary.

### *Example*

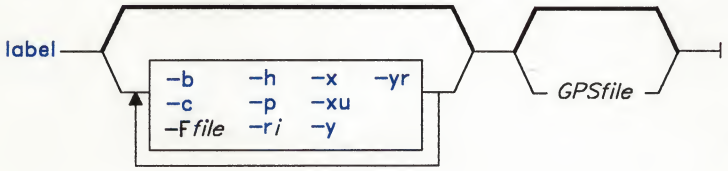
To produce a histogram:

```
hist  -r5,ya  myfile
```

This outputs the histogram described by vector `myfile` and locates it in region 5 of the **GPS** universe, with no y-axis labels.

## label

### *Syntax*




OL805541

### *Description*

The **label** command labels the axis of a **GPS** file.

### *Flags*

**-b** Assumes that the input is a bar chart.

**-c** Retains lowercase letters in labels; otherwise all letters are uppercase.

**-F***file* Specifies a label file. Each line of the file is taken as one label. Blank lines yield null labels. Either the **GPS** or the label file, but not both, can come from standard input.

**-h** Assumes that the input is a histogram.

**-p** Assumes that the input is an x-y plot. This is the default assumption.

**-r***i* Rotates labels *i* degrees. The pivot point is the first character.

**-x** Labels the x-axis. This is the default action.

**-xu** Labels the upper x-axis (the top of the plot).

**-y** Labels the y-axis.

**-yr** Labels the right y-axis (the right side of the plot).

### *Examples*

1. To label a plot:

```
label  -Flabs  A.g
```

   The file `A.g`, assumed to be an x-y plot, is labeled with labels from the file `labs`.

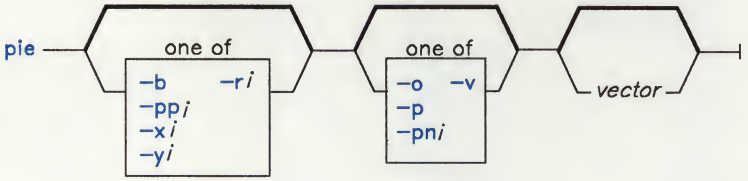
2. To label a plot from labels taken from standard input:

```
label  -yr,r-45  A.g
```

   The file `A.g` is labeled from the standard input. The labels are printed at 45 degrees below the horizontal.

## pie

### *Syntax*




OL805542

### *Description*

The **pie** command builds a pie chart. The input vector has a restricted format. Each input line represents a slice of the pies and has the following form:

[<i e f c*color*>] *value* [*label*]

with brackets indicating optional fields. The control field options have the following effects:

| | |
|---|---|
| **i** | The slice will not be drawn, though a space will be left for it. |
| **e** | The slice is "exploded" or moved away from the pie. |
| **f** | The slice is filled. The angle of fill lines depends on the color of the slice. |
| **c***color* | The slice is drawn in the specified color rather than the default black. Legal values are **b** (black), **r** (red), **g** (green), and **u** (blue). |

The pie is drawn with the value of each slice printed inside and the label printed outside.

### *Flags*

**-b** Draws pie chart with bold weight lines (the default weight is medium).

**-o** Places output values around the outside of the pie.

**-p** Expresses output values as a percentage of the total pie.

**-pn***i* Expresses output values as a percentage, but the total of the percentages equals *i* rather than 100.

**-pp***i* Draws only *i* percent of the pie.

**-r***i* Puts the pie chart in **GPS** region *i*, where *i* is between 1 and 25 inclusive (13 by default).

**-v** Does not output values.

**-x***i*
**-y***i* Positions the pie chart in the **GPS** universe with x-origin (y-origin) at *i*.

### *Example*

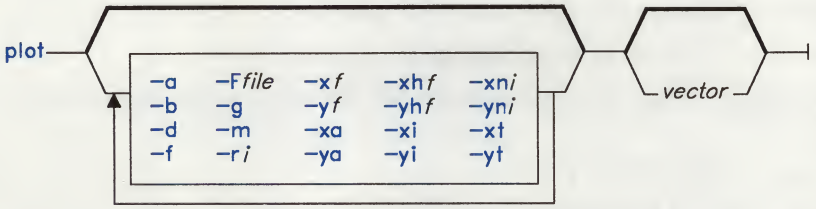
To draw a pie chart:

```
pie -pp80,pn80  chartfile
```

This draws the pie chart specified by `chartfile` in 80% of a circle and outputs the values as percentages of that total 80 percent.

## plot

### *Syntax*




OL805543

### *Description*

The **plot** command plots a graph. The input vectors contain the y values of an x-y graph. Values for the x-axis come from the file specified by **-F**. Axis scales are determined from the first vector plotted.

### *Flags*

**-a** Suppresses the axes.

**-b** Plots the graph with bold weight lines (medium is the default weight).

**-d** Does not connect plotted points (this implies **-m**).

**-f** Does not build a frame around the plot area.

**-F***file* Uses the specified file for x values; otherwise the positive integers are used. You can specify this flag more than once, causing a different set of x values to be paired with each input vector. If there are more input vectors than sets of x values, the last set applies to the remaining vectors.

**-g** Suppresses the background grid.

**-m** Marks the plotted points.

**-r***i* Puts the graph in **GPS** region *i*, where *i* is between 1 and 25 inclusive (13 by default).

**-x***f*
**-y***f* Positions the graph in the **GPS** universe with the x-origin (y-origin) at *f*.

**-xa**
**-ya** Does not label the x-axis (y-axis).

**-xh***f*
**-yh***f* Specifies the x-axis (y-axis) high boundary.

**-xl***f*
**-yl***f* Specifies the x-axis (y-axis) low boundary.

**-xn***i*
**-yn***i* Specifies the approximate number of ticks on the x-axis (y-axis).

**-xt**
**-yt** Omits the x-axis (y-axis) title.

## *Examples*

1. To plot against the positive integers:

```
plot  plotdata
```

2. To customize x- and y-axes:

```
plot  -r5,y10,xa,Fxfile  yfile
```

This plots vector `yfile` against vector `xfile`, with y-axis ticks beginning at zero, no x-axis labels printed, and the plot placed in region 5 of the **GPS** universe.

```
plot  -`hilo -oy  filea fileb`  filea  fileb
```

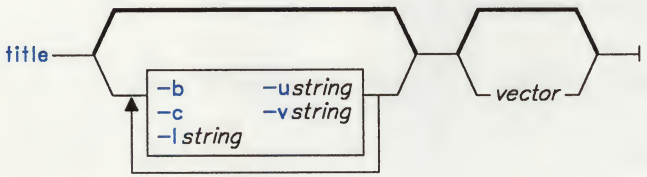
This plots vectors `filea` and `fileb` against the positive integers, with y-axis ticks going from the lowest to the highest values in the two vectors.

```
plot -Ffilea,Ffileb  filec  filed  filee
```

This plots vectors `filec` against `filea`; `filed` and `filee` against `fileb`. The y-axis scale is determined from `filec`; the x-axis scale from `filea`.

## title

### *Syntax*




OL805544

### *Description*

The **title** command prefixes a title to a vector or appends one to a **GPS** object.

### *Flags*

**-b** Makes the **GPS** title bold.

**-c** Retains lowercase letters in the title; otherwise all letters are uppercase.

**-l***string* Uses the specified string as a **GPS** lower title.

**-u***string* Uses the specified string as a **GPS** upper title.

**-v***string* Labels a vector with the specified string.

## Related Information

The following commands: "**ged**" on page 463, "**graphics**" on page 497, and "**spline**" on page 972.

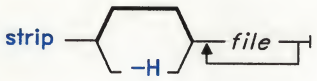
The **gps** file in *AIX Operating System Technical Reference.*

# strip

## Purpose

Removes symbol and line number information from a common object file.

## Syntax




OL805265

## Description

The **strip** command removes the symbol table and line number information from common object files, including archive libraries. Once you use this command, symbolic debugging of the file is difficult; therefore, you should normally run **strip** only on production modules that you have debugged and tested. Using **strip** reduces the file storage overhead required by an object file.

For each object module, **strip** removes all symbol table information. For each archive, **strip** removes the local symbol table information from each member.

You can restore a stripped symbol table to an archive or library file by using the **ar -s** command.

## Flag

**-H** Removes the object file header as well as all symbol table information.

## Files

/usr/tmp/strp*

## Related Information

The following commands: "**ar**" on page 55, "**as**" on page 61, "**cc**" on page 140, "**dump**" on page 366, "**ld**" on page 557, "**nm**" on page 705, and "**size**" on page 949.

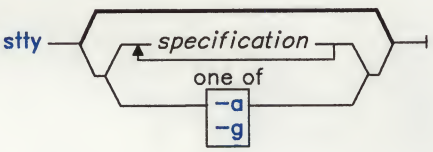
The **ar** and **a.out** files in *AIX Operating System Technical Reference.*

# stty

## Purpose

Sets, resets, or reports work station operating parameters.

## Syntax




OL805266

## Description

The **stty** command sets certain work station I/O options for the device that is the current standard input. If you run it without any *specifications*, **stty** writes to standard output information about any system adapters installed and reports the settings of certain options.

If you list any work station *specifications*, **stty** sets or resets the specified work station options.

You can find detailed information about the modes listed in the first six of the following groups in the discussion of the **termio** special facility in *AIX Operating System Technical Reference*. The last group contains options produced by combining options in the first six groups.

**Note:** The **stty** command does not make compatibility checks on any parameter combinations.

## Flags

**-a** Writes the current state of all option settings to standard output.

**-g** Writes option settings to standard output in a form usable by another **stty** command.

# Specifications

## Control Modes

The following options apply only when your work station connects to the system through an asynchronous line adapter. See **asy** in *AIX Operating System Technical Reference* for detailed information about this group.

**parenb (-parenb)** Enables (disables) parity generation and detection.

**parodd (-parodd)** Selects odd (even) parity.

**cs5 cs6 cs7 cs8** Selects character size.

**Note:** See **termio** in *AIX Operating System Technical Reference* for additional information on character size.

**0** Hangs up phone line immediately.

**50 75 110 134 150 300 600 12001800 2400 4800 9600 19200 19.2 38400 38.4 exta extb**
Sets the work station speed to the specified number of bits per second (**exta**, 19200, and 19.2 are synonyms; extb, 38400, and 38.4 are synonyms). Regardless of the baud rate, the software requires that terminals generate and support the ASCII character set.

**hupcl (-hupcl)**

**hup (-hup)** Hangs up (does not hang up) dial-up connection on the last close.

**cstopb (-cstopb)** Selects 2 (1) stop bits per character.

The next two options apply to all work stations, regardless of the line adapter:

**cread (-cread)** Enables (disables) the receiver.

**clocal (-clocal)** Assumes a line without (with) modem control. The status of **clocal** is displayed when you use the **-a** flag. You cannot change this status with the **stty** command.

## Input Modes

**ignbrk (-ignbrk)** Ignores (does not ignore) **BREAK** on input.

**brkint (-brkint)** Signals (does not signal) **INTR** on break.

**ignpar (-ignpar)** Ignores (does not ignore) parity errors.

**parmrk (-parmrk)**
Marks (does not mark) parity errors.

| | |
|---|---|
| **inpck (-inpck)** | Enables (disables) input parity checking. |
| **istrip (-istrip)** | Strips (does not strip) input characters to 7 bits. |
| **inlcr (-inlcr)** | Maps (does not map) **NL** to **CR** on input. |
| **igncr (-igncr)** | Ignores (does not ignore) **CR** on input. |
| **icrnl (-icrnl)** | Maps (does not map) **CR** to **NL** on input. |
| **iuclc (-iuclc)** | Maps (does not map) uppercase alphabetic characters to lowercase. |
| **ixon (-ixon)** | Enables (disables) START/STOP output control. Once START/STOP output control has been enabled, you can pause output to the work station by pressing **Ctrl-S** and resume output by pressing **Ctrl-Q**. |
| **ixany (-ixany)** | Allows any character (only **Ctrl-Q**) to restart output. |
| **ixoff (-ixoff)** | Sends (does not send) START/STOP characters when the input queue is nearly empty/full. |

## Output Modes

| | |
|---|---|
| **opost (-opost)** | Processes output (does not process output; that is, it ignores all other output options). |
| **olcuc (-olcuc)** | Maps (does not map) lowercase alphabetic characters to uppercase on output. |
| **onlcr (-onlcr)** | Maps (does not map) **NL** characters to **CR-NL** characters. |
| **ocrnl (-ocrnl)** | Maps (does not map) **CR-NL** characters to **NL** characters. |
| **onocr (-onocr)** | Does not (does) output **CR** characters at column zero. |
| **onlret (-onlret)** | On the terminal, **NL** performs (does not perform) the **CR** function. |
| **ofill (-ofill)** | Uses fill characters (uses timing) for delays. |
| **ofdel (-ofdel)** | Uses **DEL (NUL)** characters for fill characters. |
| **cr0 cr1 cr2 cr3** | Selects style of delay for **CR** characters. |
| **nl0 nl1** | Selects style of delay for **NL** characters. |

**tab0 tab1 tab2 tab3**
Selects style of delay for horizontal tabs.

**bs0 bs1** Selects style of delay for backspaces.

**ff0 ff1** Selects style of delay for form feeds.

**vt0 vt1** Selects style of delay for vertical tabs.

## Local Modes

**isig (-isig)** Enables (disables) the checking of characters against the special control characters **INTR** and **QUIT**.

**icanon (-icanon)** Enables (disables) ***canonical input*** (canonical input allows input-line editing with the **ERASE** and **KILL** characters).

**xcase (-xcase)** Echoes (does not echo) uppercase characters on input, and displays uppercase characters on output with a preceding \ (backslash).

**echo (-echo)** Echoes (does not echo) every character typed.

**echoe (-echoe)** Echoes (does not echo) the **ERASE** character as the backspace-space-backspace string.

**Note:** This mode does not keep track of column position, so you may get unexpected results when erasing tabs, escape sequences, and the like.

**echok (-echok)** Echoes (does not echo) a **NL** character after a **KILL** character.

**lfkc (-lfkc)** Functions the same as **echok**. This is an obsolete mode.

**echonl (-echonl)** Echoes (does not echo) the **NL** character.

**noflsh (-noflsh)** Does not clear (does clear) buffers after **INTR** or **QUIT**.

## Control Assignments

*control-character c* Set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **eof**, **eol**, **min**, or **time**. (Use **min** and **time** with **-icanon**.) If *c* is in the form \^*c* (backslash circumflex *c*), then its value is the corresponding **Ctrl** character. A \^**?** (backslash circumflex question mark) is interpreted as **DEL**. A \^- (backslash circumflex minus) is interpreted as undefined.

**enhedit (-enhedit)**
Enters (leaves) the enhanced line editing discipline (see the **termio** special facility in *AIX Operating System Technical Reference*).

**Note:** When Japanese Language Support is installed, **enhedit** is not supported.

**ascedit (-ascedit)**
Enters (leaves) the ASCII keyboard mode for **dosedit**.

**line** *i* Sets the line discipline. *i* can be either 0 or 1. `stty line 0` is the same as `stty -enhedit`. `stty line 1` is the same as `stty enhedit`.

## Screen Length

**page (-page)** Pauses (does not pause) during output after each screen displayed. Typing any character during the pause causes output to resume. Typing a space during the pause causes output to continue uninterrupted until the next command is entered.

**length** *n* Sets screen length to *n* lines, where *n* is an integer from 1 through 255. An automatic pause in output occurs after *n* lines if **page** is enabled.

## Combination Modes

**evenp | parity** Enables **parenb** and **cs7**.

**oddp** Enables **parenb**, **cs7**, and **parodd**.

**-parity, -evenp, -oddp**
Disables **parenb** and sets **cs8**.

**raw (-raw | cooked)** Enables (disables) ***raw*** input and output (no **ERASE**, **KILL**, **INTR**, **QUIT**, **EOT**, or output processing).

| | |
|---|---|
| **nl (-nl)** | Unsets (sets) **icrnl** and **onlcr**. Specifying **-nl** sets **icrnl** and **onlcr** and also unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**. |
| **lcase (-lcase)**<br>**LCASE (-LCASE)** | Sets **xcase**, **iuclc**, and **olcuc**. (Used for work stations with uppercase characters only.) |
| **tabs (-tabs ¦ tab3)** | Preserve tabs (expand to spaces) when printing. |
| **ek** | Sets **ERASE** and **KILL** characters to **Ctrl-H** and **Ctrl-U**, respectively. |
| **sane** | Resets parameters to "reasonable" values. |
| *term* | Sets all parameters according to work station type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**. |

### Terminal Mapping

| | |
|---|---|
| **dmap** *mapname* | Deactivates a loaded map. Any user can deactivate a map. |
| **imap** *mapname* | Loads and activates **/etc/nls/termmap/***mapname***.in** as the terminal input map. If no *mapname* is defined, **imap** activates the previously specified map.<br><br>**Note:** You must be a superuser or a member of the system group to load a map. Any user can activate a loaded map with **imap**. |
| **omap** *mapname* | Loads and activates **/etc/nls/termmap/***mapname***.out** as the terminal output map. If no *mapname* is defined, **omap** activates the previously specified map.<br><br>**Note:** You must be a superuser or a member of the system group to load a map. Any user can activate a loaded map with **omap**. |

## Examples

1. To display a short listing of your work station configuration:

   ```
   stty
   ```

   This lists settings that differ from the defaults.

2. To display a full listing of your work station configuration:

   ```
   stty  -a
   ```

3. To enable a key sequence that stops listings from scrolling off the screen:

   ```
   stty  ixon  ixany
   ```

   This sets **ixon** mode, which lets you stop runaway listings by pressing **Ctrl-S**. The `ixany` parameter allows you to resume the listing by pressing any key. The normal

work station configuration includes `ixon` and `-ixany`, which allows you to stop a listing with **Ctrl-S**, but only **Ctrl-Q** will restart it.

4. To prevent all listings from scrolling off the screen:

```
stty  page  length  24
```

This sets `page` mode with a page (screen) length of `24` lines. When a listing is more than 24 lines long, the system pauses after each page. It beeps, reminding you to press any key (except the **Spacebar**) to view the next page. Press the **Spacebar** to let the rest of the listing scroll off the screen and get to the end. Paging then resumes with the next listing.

5. To reset the configuration after it has been messed up:

**Ctrl-J** `stty  sane  echo  -tabs` **Ctrl-J**

Sometimes the information displayed on the screen may look strange, or the system won't respond when you press the **Enter** key. This can happen when you use **stty** with parameters that are incompatible or that do things you don't understand. It can also happen when a screen-oriented text editor ends abnormally and doesn't have a chance to reset the work station configuration.

Entering `stty sane` sets a reasonable configuration, but it may differ slightly from your normal configuration. That is why this example also includes two commonly used parameters, `echo` (erase characters as you backspace over them) and `-tabs` (expand tab characters to spaces on the display screen).

Press **Ctrl-J** before and after the command instead of **Enter**. The system usually recognizes **Ctrl-J** when the parameters that control the **Enter** key processing are messed up.

6. To save and restore the work station's configuration:

```
OLDCONFIG=`stty  -g`          #  save  configuration
stty  -echo                    #  do not  display  password
echo  "Enter password:  \c"
read  PASSWD                   #  get  the  password
stty  $OLDCONFIG               #  restore  configuration
```

This saves the work station's configuration, turns off echoing, reads a password, and restores the original configuration. The ` . . . ` (grave accents) in the first command tell the shell to insert the standard output of `stty -g` into the `OLDCONFIG= . . .` command. This is called ***command substitution***. For more information, see "Command Substitution" on page 925.

The `stty -echo` turns off echoing, which means that the password does not appear on the screen when you type it at the keyboard. This has nothing to do with the **echo** command, which displays a message on the screen.

## Related Information

The following command: "**tabs**" on page 1041.

The **ioctl** system call and the **terminfo** and **config** files in *AIX Operating System Technical Reference.*

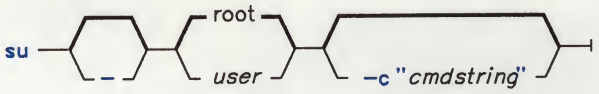
The discussion of **stty** and "Overview of International Character Support" in *IBM RT Managing the AIX Operating System.*

# su

## Purpose

Obtains the privileges of another user, including superuser authority.

## Syntax




OL805267

## Description

The **su** command runs a shell and lets you operate there with the privileges of the specified *user* (by default `root`). If you are not already operating with superuser authority, **su** prompts for the password associated with *user* before granting you these privileges. Upon successful authentication, **su** will set the audit classes for the new shell to those specified in **/etc/security/passwd**.

If the terminal is in a trusted state, the shell will be **tsh** and the prompt will be set to `tsh`.

If you use **su** to become the superuser (*user* is `root`), **su** sets the **PATH** variable to **/bin:/etc:/usr/bin** and changes the prompt to # (pound sign). Notice that this **PATH** does not include the current directory.

To restore your normal privileges, press **END OF FILE (Ctrl-D)**. This action ends the shell called by **su** and returns you to the previous shell and ID.

If you need to run only one command as *user*, you can run the desired command by including it (along with any of its associated flags) on the command line as an argument to the shell **-c** flag (see "**sh**" on page 913 for a description of this flag). In this case, **su** calls **sh** to run the command and then exits automatically.

Each time someone uses **su** to become the superuser, **su** writes a record in the file **/usr/adm/sulog**, creating this file if necessary.

**Note:** If the **-c** flag is not specified, **su** execs the shell listed in the shell field of the **/etc/passwd** file. If the **-c** flag is specified, **su** ignores the **passwd** file entry and runs **/bin/sh**. All exported environment variables are available unless you use the - flag when you call **su**.

## Flags

Creates the user's login environment by calling the new shell as a ***login shell*** (see "**sh**" on page 913). It reads the system **profile** file and the *user*'s **$HOME/.profile** file. The environment variables **NLLDATE** and **NLTIME** control the appearance of the date and time. The **TERM** and **TZ** variables are an exception. They are preserved at their current values. These variables are normally set by **init** or **getty** prior to login; as a result, **su** handles them differently.

**Note:** This flag modifies the environment of the current shell only if the optional program named in the shell field of the **passwd** file is a program like **sh**.

**-c** "*cmdstring*" Runs the **/bin/sh** shell, processes the specified *command*, and then exits the shell. This flag causes **su** to ignore the shell specified in the **passwd** file.

## Examples

1. To obtain superuser authority:

   ```
   su
   ```

   This runs a subshell with the effective user ID and privileges of user `root`. The **su** command asks for a password, as if you were logging in as `root`. Now the commands you run have superuser authority. Press **END OF FILE (Ctrl-D)** to end the subshell and return to your original shell session and privileges.

2. To obtain `jim`'s privileges:

   ```
   su  jim
   ```

   This runs a subshell with the effective user ID and privileges of `jim`.

3. To set up the environment as if you had logged in as `jim`:

   ```
   su  -  jim
   ```

   This runs a subshell with the effective user ID and privileges of `jim`. The - (minus) causes the shell variable **LOGNAME** to be set to `jim`, **HOME** to be set to the path name of `jim`'s home directory, and `jim`'s **$HOME/.profile** shell procedure file to be run before prompting for the first shell command.

4. To run a single command with superuser authority:

```
su  root  -c "backup  -9  -u"
```

This runs the shell command `backup -9 -u` with superuser authority (if you know the password assigned to `root`).

## Related Information

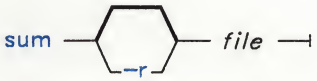
The following command: "**sh**" on page 913.

# sum

## Purpose

Displays the checksum and block count of a file.

## Syntax




OL805268

## Description

The **sum** command reads *file* and calculates a 16-bit checksum for the *file* and the number of blocks in the file. The checksum and number of blocks are written to standard output. The **sum** command is generally used to determine if a file that has been copied or communicated over transmission lines is an exact copy of the original.

## Flag

**-r** Uses an alternate algorithm to compute the checksum (rigorous byte-by-byte computation rather than the default word by word computation).

## Example

To display the checksum of, and the number of blocks in `datafile`:

```
sum  datafile
```

If the checksum of `datafile` is 1605 and if the file contains 3 blocks, then **sum** displays:

```
1605 3 datafile
```

## Related Information

The following command: "**wc**" on page 1211.

# sync

## Purpose

Updates the superblock and writes buffered files to the fixed disk.

## Syntax


OL805221

## Description

The **sync** command runs the **sync** system primitive. If you have to stop the system, you must run **sync** to ensure file system integrity. **sync** writes all unwritten system buffers to disk. This includes modified superblocks, modified i-nodes, delayed block I/O, and read-write mapped files.

**Note:** The writing, although scheduled, is not necessarily complete upon return from the **sync** system call.

## Related Information

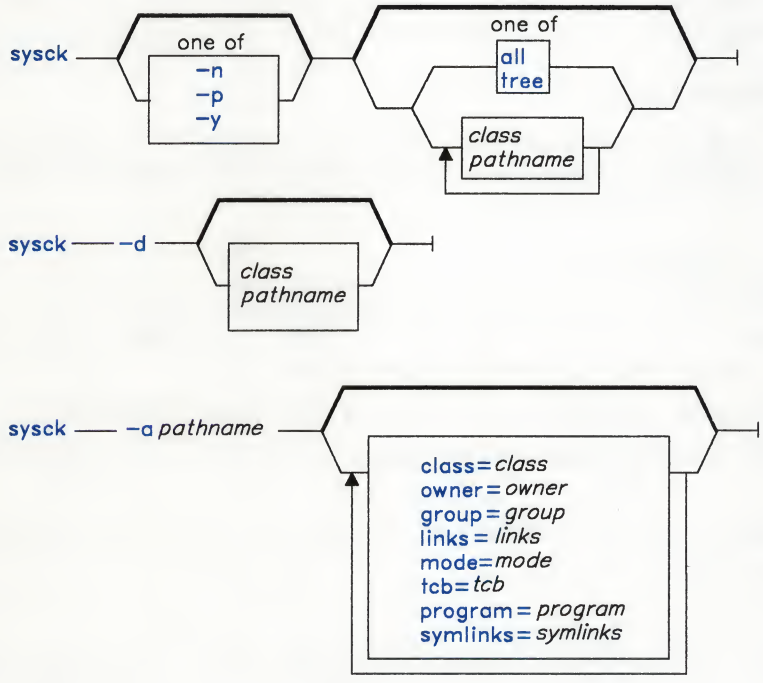
The **sync** system call in *AIX Operating System Technical Reference*.

# sysck

## Purpose

Verifies the secure system state

## Syntax




OL805503

## Description

The **sysck** command checks the installation of files relevant to security. Each such file must have a corresponding stanza in **/etc/security/sysck.cfg**. The **sysck** command also adds or deletes file descriptions from **/etc/security/sysck.cfg**.

When invoked with no flags and with no arguments, the **sysck** command prints a synopsis of its usage.

## Adding a File Description

When invoked with the **-a** flag, the **sysck** command adds or modifies a stanza in the **/etc/security/sysck.cfg** file. The added or modified stanza describes the file specified by *pathname*. The file attributes are taken from:

- The old stanza for the file (if it exists).
- The file (which must exist).
- The *attribute*=*value* arguments on the command line (if there are any).

You may specify that an attribute is not to appear in the stanza by specifying an empty string as the value for the attribute. See the discussion of the **sysck.cfg** file in *AIX Operating System Technical Reference* for a description of the attributes recorded in **/etc/security/sysck.cfg**.

## Deleting a File Description

When invoked with the **-d** flag, the **sysck** command deletes the specified stanzas from **/etc/security/sysck.cfg**. A stanza may be deleted by specifying its *pathname* or by specifying the value of its **class** attribute.

## Checking a File Description

When invoked with neither the **-a** nor **-d** flags, the **sysck** command checks the installation of files described in **/etc/security/sysck.cfg**. Any inconsistency can be fixed by changing the file to match the description in **/etc/security/sysck.cfg**.

The arguments to the **sysck** command specify which stanzas are to be checked:

*pathname* Specifies the name of a stanza. This must be an absolute path name (i.e., begin with a /).

*class* Specifies all stanzas whose **class** attribute has the value *class*. Note that a class name cannot begin with a /.

**all** Specifies all stanzas.

**tree** Specifies all stanzas. When **tree** is specified, the **sysck** command not only checks the installation of files described in **/etc/security/sysck.cfg**, but it also checks that there are no other files in the file system which should be described by **/etc/security/sysck.cfg**.

The following checks are performed on each file *not* described by a stanza in **/etc/security/sysck.cfg**:

- If the file is a hard link to a file described in **/etc/security/sysck.cfg**, but is not listed in the value of the **links** attribute, the link is removed.
- If the file is a symbolic link to a file described in **/etc/security/sysck.cfg**, but is not listed in the value of the **symlinks** attribute, the symbolic link is removed.

- If the file is a regular file, the file owner ID is 0, and the file mode has the **S_ISUID** bit set, the **S_ISUID** bit is cleared.
- If the file is a regular file, the file group ID is 0, and the file mode has the **S_ISGID** and **S_IXGRP** bits set, the **S_ISGID** bit is cleared.
- If the file has the **tcb** attribute set, the **tcb** attribute is cleared.
- If the file is a block or character device, all permission bits of the file are set to 0.

The action to be taken when a configuration error is found is determined by any **-n**, **-p**, or **-y** flag supplied to the **sysck** command.

The following checks are performed if the stanza has the specified attribute. If an inconsistency is found, **sysck** performs the action specified by the flag supplied with the **sysck** command.

**class** The value of this attribute is the name of a group of stanzas. The **sysck** command performs no checks based on the value of this attribute. The **class** attribute is used to specify a group of stanzas to be checked by the **sysck** command.

**owner** The value of this attribute is a decimal user ID, or the login name of a user in **/etc/passwd**. The file owner ID should be this user ID.

**group** The value of this attribute is a decimal group ID, or the name of a group in **/etc/group**. The file group ID should be this group ID.

**mode** The value of this attribute can be an octal numeral, or a string in the form `-r-Sr-x--T`. The file mode should be this value.

**links** The value of this attribute is a comma-separated list of additional links to the file.

**symlinks** The value of this attribute is a comma-separated list of symbolic links to the file. Note that **sysck** will find additional symbolic links only if given the **tree** argument.

**tcb** The value of this attribute is **true** or **false**. If **true**, the file should be flagged as part of the trusted computing base.

**program** The value of this attribute is the full path name of a program that performs additional consistency checks on the file. Arguments to the program also appear with this attribute. Note that the value of an attribute must be enclosed in double quotes if it has embedded spaces. The **sysck** command will execute this program with any arguments preceded by the same flags (**-n**, **-p**, or **-y**) as were provided on the **sysck** command line.

## Consistency Checks

Two consistency check programs are supplied: **pwdchk** and **grpchk**.

### /etc/security/pwdchk

The **pwdchk** program checks the **/etc/passwd** and **/etc/security/passwd** files for internal and mutual consistency. Consistency checks are not applied to lines in **/etc/passwd** with start with - (minus) or + (plus).

The internal consistency checks for **/etc/passwd** include the following:

- Each line must have seven colon-separated fields. Any malformed entry is reported but not corrected.
- The *password* field of each entry must be ! (exclamation point). An invalid entry is reported and set to ! (exclamation point); if the *password* file is 13 or more characters, the leading 13 characters become the value of the **password** attribute in **/etc/security/passwd**.
- The *uid* field of each entry must be decimal numeral. An invalid entry is reported and the value of the **restrictions** attribute in **/etc/security/passwd** is set to **nouse**.
- The *gid* field of each entry must be decimal numeral which corresponds to a group described in **/etc/group**. An invalid entry is reported and the value of the **restrictions** attribute in **/etc/security/passwd** is set to **nouse**.
- The *directory* field must be **NULL** or a full path name. An invalid entry is reported but not corrected.
- The *shell* fields must be **NULL** or a full path name. An invalid entry is reported but not corrected.

The mutual consistency checks for **/etc/passwd** and **/etc/security/passwd** include the following:

- For every entry in **/etc/passwd**, there must be a stanza with the same user name in **/etc/security/passwd**. A stanza is created in **/etc/security/passwd** for any extra entries in **/etc/passwd**.
- For every entry in **/etc/security/passwd**, there must be a corresponding entry in **/etc/passwd**. Extra entries in **/etc/security/passwd** are reported and deleted.
- Each entry in **/etc/passwd** must have a unique user ID, unless **/etc/security/passwd** has a stanza for this user in which the **restrictions** attribute is **none**. Only one login account is allowed for each unique user ID unless specifically allowed by `restrictions = none` in **etc/security/passwd**. Subsequent entries with the same user ID as a previous entry are reported and the value of the **restrictions** attribute is set to **nologin**.

### /etc/security/grpchk

The **grpchk** program checks the **/etc/group** and **/etc/security/group** files for internal and mutual consistency.

The internal consistency checks for **/etc/group** include the following:

- Each line must have three colon-separated fields. Any malformed entry is reported, but not corrected.
- Each group name must be unique. A duplicate entry is reported, but not corrected.
- The *gid* field of each entry must be unique. An invalid entry is reported, but not corrected.
- Each user listed as a member of the group must have an entry in **/etc/passwd**. An unknown user name is reported, but not corrected.

The mutual consistency checks for **/etc/group** and **/etc/security/group** include the following:

- For every entry in **/etc/group**, there must be a stanza with the same group name in **/etc/security/group**. A stanza is created in **/etc/security/group** for any extra entries in **/etc/group**.
- For every entry in **/etc/security/group**, there must be a stanza with the same group name in **/etc/group**. A stanza is created in **/etc/group** for any extra entries in **/etc/security/group**.

---

**Japanese Language Support Information**

If Japanese Language Support is installed on your system, this command is not available.

---

## Flags

**-a** Adds (to **/etc/security/sysck.cfg**) the description of a file.

**-d** Deletes (from **/etc/security/sysck.cfg**) the description of the files specified by the arguments.

**-n** Checks the installation of the files specified by the arguments. Any inconsistencies are reported. No changes will be made to either the file or to **/etc/security/sysck.cfg**.

**-p** Checks the installation of the files specified by the arguments. Any inconsistencies are **not** reported. Changes are made to files so they match the description in **/etc/security/sysck.cfg**.

**-y** Checks the installation of the files specified by the arguments. Any inconsistencies are reported. Changes are made to files so they match the description in **/etc/security/sysck.cfg**.

## Example

The following command assures all files described by **/etc/security/sysck.cfg** are properly installed:

```
sysck -y all
```

The following command can be used to construct an **/etc/security/sysck.cfg** file which reflects the current state of the system:

```
sysck tree
```

## Files

| | |
|---|---|
| /etc/security/sysck.cfg | Defines installation assertions for security relevant files and directories. |

## Related Information

The following file format: **sysck.cfg** in *AIX Operating System Technical Reference.*

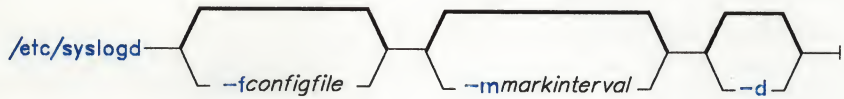
The discussion of the trusted computing base in *Managing the AIX Operating System.*

# syslogd

## Purpose

Reads and logs messages.

## Syntax




AJ2FL148

## Description

The **syslogd** command reads and logs messages into a set of files described by the configuration file **/etc/syslog.config**. This daemon configures itself when it starts up and whenever it receives a hangup signal.

Each message read by **syslogd** is one line. A message can contain a priority code (marked by a number in < > brackets at the beginning of the line) and message text. Priorities are defined in **sys/syslog.h**. The **syslogd** command reads from the AIX domain socket **/dev/log** or from an Internet domain socket specified in **/etc/services**.

Each line in the **syslogd** configuration file must consist of two parts:

- A selector to determine the message priorities to which the line applies
- An action.

The two fields must be separated by one or more tabs. Here is an example of the line in a configuration file:

```
mail.info;*.notice        /usr/spool/adm/syslog
```

The first part, the selector, is semicolon-separated list of priority specifiers. Each priority specifier consists of a facility describing the part of the system that generated the message, a . (period), and a level indicating the severity of the message. Symbolic names may be used and an * (asterisk) specifies all facilities. All messages of the specified level or higher (greater severity) are selected. In the previous example, **syslogd** selects the `mail` facility at the `info` level (or higher) and all facilities at the `notice` level (or higher).

More than one facility may be selected using commas to separate them. For example:

```
*.emerg;mail,daemon.crit
```

selects all facilities at the `emerg` level (or higher) and the `mail` and `daemon` facilities at the `crit` level (or higher).

Known facilities and levels recognized by **syslogd** are those listed under **syslog** in the *AIX Operating System Technical Reference*. When you specify the name of a facility or level in a **syslogd** configuration file, omit the **LOG_** prefix used by **syslog** in the name. For example, **syslog** lists **LOG_DEBUG** as the lowest level. To specify this level in a **syslogd** configuration file, specify `debug`.

In addition to these facilities, there is a **mark** facility. This facility has messages at priority **info** sent to it every 20 minutes. You can change the mark time interval with the **-m** flag. The **mark** facility is not enabled by a facility field containing an asterisk; you must explicitly enable it. For example:

```
kern,mark.debug
```

logs kernel messages and 20-minute marks of `debug` level (or higher).

The level none may be used to disable a particular facility. For example:

```
*.debug;mail.none
```

logs all messages except mail messages.

The second part of each line, the action, describes where the message is to be logged if the line is selected. There are four forms:

- A file name beginning with a leading / (Selected messages are appended to this file)
- A host name preceded by a @ (Selected messages are forwarded to **syslogd** on the named host)
- A comma-separated list of users (Selected messages are written to those users, if they are logged in)
- An * (Selected messages are written to all logged-in users).

For example:

```
*.crit                  /usr/adm/critical
kern.err                @nick
*.alert                 bobbi,kristi
*.emerg                 *
```

logs critical (or higher) messages into `/usr/adm/critical`, forwards kernel messages of error severity (or higher) to **syslogd** on the host `nick`, informs the users `bobbi` and `kristi` of any alert (or higher) messages, and informs all logged-in users of any emergency messages.

Blank lines and lines beginning with # are ignored.